DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Evaluation of cross-platform technology Flutter from the user's perspective

ADIBBIN HAIDER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Evaluation of cross-platform technology Flutter from the user's perspective

ADIBBIN HAIDER

Master in Computer Science
Date: October 3, 2021
Supervisor: Somayeh Aghanavesi
Examiner: Elena Troubitsyna
School of Electrical Engineering and Computer Science
Swedish title: Utvärdering av multiplattformsteknologin Flutter från ett användarcentrerat perspektiv

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, multiple sample applications were developed. For evaluating the user perception between Flutter and native applications, a user study was conducted using a UEQ. 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 sample 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.

Sammanfattning

Syftet med avhandlingen var att utvärdera multiplattformsteknologin Flutter utifrån ett användarperspektiv. De viktigaste aspekter som undersöktes var användarupplevd prestanda som starttid och applikationsstorlek. Dessutom var användaruppfattning en nyckelfunktion som undersöktes. För att utvärdera plattformstekniken Flutter utvecklades flera exempelapplikationer. För att jämföra användarnas uppfattning mellan Flutter och native applikationer genomfördes en användarstudie med en UEQ. Resultaten indikerade att om applikationens prestanda är avgörande för användarna är en Flutter-applikation sannolikt inte lämplig. Användaruppfattningsstudien visade att det inte fanns några signifikanta skillnader mellan de utvecklade provapplikationerna. Inga signifikanta skillnader fanns mellan Flutter-applikationerna och den ursprungliga applikationen för varken Android- eller iOS-plattformen. Således kan en Flutter-applikation vara ett lämpligt alternativ för en inbyggd Android-applikation ur användarens perspektiv.

Contents

1 Introduction **1**

1.1 Introduction . . . 1

1.2 Motivation . . . 2

1.3 Research questions . . . 2

1.4 Aim . . . 3

1.5 Delimitations . . . 3

1.6 Related work . . . 3

1.6.1 Sample applications . . . 4

1.6.2 User experience and user interface . . . 5

1.6.3 Performance . . . 5

2 Background **7**

2.1 Cross-platform applications . . . 7

2.1.1 Flutter . . . 8

2.2 Native mobile applications . . . 12

2.2.1 Development . . . 13

2.3 Model-View-ViewModel . . . 13

2.4 t-Test . . . 15

2.5 User experience questionnaire . . . 16

3 Method **19**

3.1 Developing sample applications . . . 19

3.1.1 Complexity . . . 19

3.1.2 Design pattern . . . 20

3.1.3 User experience and user interface . . . 21

3.1.4 Developed applications . . . 21

3.2 Performance metrics . . . 25

3.2.1 Size . . . 25

3.2.2 Startup time . . . 26

3.3 User perception study 26
3.3.1 Why UEQ? 26
3.3.2 Conducting the UEQ 27

4 Result 30
4.1 User perceived performance 30
4.1.1 Size 30
4.1.2 Startup time 31
4.2 User perception study 33
4.2.1 Android 33
4.2.2 iOS 34

5 Discussion 36
5.1 User perceived performance 36
5.1.1 Flutter overhead 36
5.1.2 Size 37
5.1.3 Startup time 38
5.2 User perception study 39
5.2.1 Method 40
5.3 Sample applications 40
5.4 User's perspective 42
5.5 Design pattern 42
5.6 Sources of error 43
5.7 Sustainability & Ethical Aspects 43

6 Conclusion 44

7 Future work 46

Bibliography 47

A User Experience Questionnaire 52

Chapter 1

Introduction

This chapter presents the current environment of mobile application development and is followed by presenting the cross-platform technology Flutter. Thenceforth, the motivation behind this thesis is explained and research questions are defined. Finally, the thesis delimitations are outlined and relevant related work is presented.

1.1 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 sample applications. 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 sample applications. Additionally, the performance will be measured while the participants in the study use the applications.

1.2 Motivation

Research to date shows that Flutter has not been investigated or evaluated thoroughly within academia. No previous studies from the user's perspective have been found to date. The perception of applications is crucial for both users and developers. Users want to use an application that satisfies their needs and meet their expectations [11]. Thus, developers have the incentive to build applications that allow individuals to use their applications to keep them satisfied and meet their expectations. Therefore, developers might be interested to know whether Flutter developed applications are an alternative from the user's perspective. Similar research exists for other cross-platform solutions. However, none for Flutter. Hence, this thesis has significant news value and lays a foundation for further work regarding Flutter.

1.3 Research questions

The general research question of this thesis is: can a Flutter developed application be a viable alternative for native application from the user's perspective?

To give a conclusive answer to the research question, it is broken down into three smaller questions. Thus, the aim is to answer the following questions as well:

- What is the effect on user-perceived performance by a Flutter developed application?
- Is it possible to achieve the same look and feel as a native application using Flutter?
- Which type of applications do users prefer, native or Flutter?

1.4 Aim

This thesis aims to answer the research question, *can a Flutter developed application be a viable alternative for native application from a user-centered perspective?* To tackle this question sample applications will be developed using Flutter and natively for Android and iOS. Thereafter, the applications will be evaluated and compared using a user experience questionnaire.

The comprehensive aim of this thesis is to help developers make the right decision. Dependent on the result of this thesis, it may encourage or discourage developers to build applications using Flutter.

1.5 Delimitations

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.

1.6 Related work

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 [12] 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 [12] 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 was deemed 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. [15]. 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.

1.6.1 Sample applications

Numerous studies based their research on developing one or more sample applications [16, 17, 18, 19, 12]. The sample applications among the studies were different from each other and no consistency on functionalities or design was found. The only similarities among the application were simplicity. The applications were simple to use and have limited functionalities. However, even though they were limited, most of them were complex enough to have at least two views.

The foundation of this thesis will also be based on developing sample applications. One native application for each of the target platforms. One cross-platform application developed using the Flutter UI toolkit. The sample application will be simple to use and have limited functionalities.

1.6.2 User experience and user interface

Various studies about cross-platform applications have also researched the user experience (UX) or user interface (UI) of the cross-platform applications [16, 20, 21, 22, 23]. One study by Angulo and Ferre [24] conducted their research specifically about the UX for cross-platform application in comparison to native applications.

All of the previous studies seemed to reach the same conclusion. Cross-platform applications tended to have an inferior UI/UX. It is possible to achieve the same UI/UX with cross-platform applications as a native application. However, it is often more challenging for the developer and thus often tends to lead to inferior UI/UX.

One study [12] specifically studied the look and feel for Android and iOS. They developed native sample applications and a cross-platform sample application using React Native. By conducting a user experience questionnaire, they concluded that they were no significant difference for the users regarding the look and feel.

If it is possible to achieve the same UI/UX or look and feel in previous cross-platform solutions, the new toolkit Flutter should be able to provide the same results. Similar to the study by Hansson and Vidhall [12], the research for the thesis will partially be based on conducting a user experience questionnaire.

1.6.3 Performance

Multiple studies [17, 12, 5] has been focusing on researching the performance of applications developed using cross-platform solutions. More general research [16, 20, 21, 22, 25] in the area of mobile cross-platform platforms, has also discussed the performance between native applications and cross-platform applications. The overall conclusion seems to be that native applications tend to perform better than cross-platform developed applications.

In research by Mercado, Munaiah, and Meneely [5] based on analyzing customer reviews for Android and iOS applications, users had a more positive perception of native applications in aspects of performance. The same study found when Facebook changed from developing non-natively to natively, the density of the user complaints regarding performance reduced substantially.

Evert [17] studied the performance difference between native applications and cross-platform applications developed using Kotlin Multiplatform. For Android, the cross-platform application was approximately 13% larger with a longer build -and startup time compared to the native application. For iOS, the cross-platform application was approximately 18% larger with a smaller build time. No indication of a significant difference between startup times for the different applications was found.

In a study by Hansson and Vidhall [12], two native sample applications were developed and one sample application was developed using the cross-platform framework React Native. The native applications performed better in regards to CPU usage, memory usage, frames per second, and response time. Additionally, the native applications were 5-15 MB smaller than the React Native application.

There is plenty of research on cross-platform application performance, and all of them conclude the same thing. Native applications have better performance than cross-platform applications. However, all of the previous research was conducted before the existence of Flutter. It exists no data on how Flutter developed applications perform. This further reinforces the importance of this thesis and the evaluation of the toolkit.

Additionally, less research has been conducted regarding the user-perceived performance or what the user experiences regarding performance. The study by Mercado, Munaiah, and Meneely [5] touched upon this area, but no more significant studies could be found. This further reinforces this thesis aim.

Chapter 2

Background

This chapter presents the underlying theory and evaluation methods used for the thesis. Starting by briefly describing the concept of cross-platform and then how Flutter works. Thenceforth, a brief description of native applications and a design pattern named Model-View-ViewModel is presented. The chapter finishes by describing the evaluation method User Experience Questionnaire.

2.1 Cross-platform applications

When developing a mobile application, the aim is likely to target as many users as possible by providing the same application on different platforms. There are two main approaches to this. Developing applications for every desired platform, or build an application once and make them run on multiple platforms. The first alternative can waste development time in learning and become familiar with different platforms development environments. This problem leads to the concept of cross-platform mobile development solutions. The goal of cross-platform solutions is to develop an application once and run it on multiple platforms. Hence, the saying *Write once, run anywhere*. [15]

Cross-platform solutions often work by creating an abstraction layer over the platform. The abstraction layer usually makes use of the platform natives libraries, such as the UI libraries. Attempting to keep a consistent look and feel for the selected platform. The cross-platform solutions application code is often written in an interpreted language, such as JavaScript. The code must interact with the underlying native platform libraries to display the UI.[26][1]

[1]Flutter architectural overview, `https://flutter.dev/docs/resources/`

2.1.1 Flutter

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[6]. 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.[7]

`architectural-overview`, Accessed 2021-02-08

[2]Flutter - Beautiful native apps in record time, `https://flutter.dev/`, Accessed 2021-02-08

[3]Web support for Flutter, `https://flutter.dev/web`, Accessed 2021-02-08

[4]Dekstop support for Flutter, `https://flutter.dev/desktop`, Accessed 2021-02-08

[5]Start thinking declaratively, `https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative`, , Accessed 2021-02-08

[6]Ibid.

[7]Flutter architectural overview, `https://flutter.dev/docs/resources/architectural-overview`, Accessed 2021-02-08

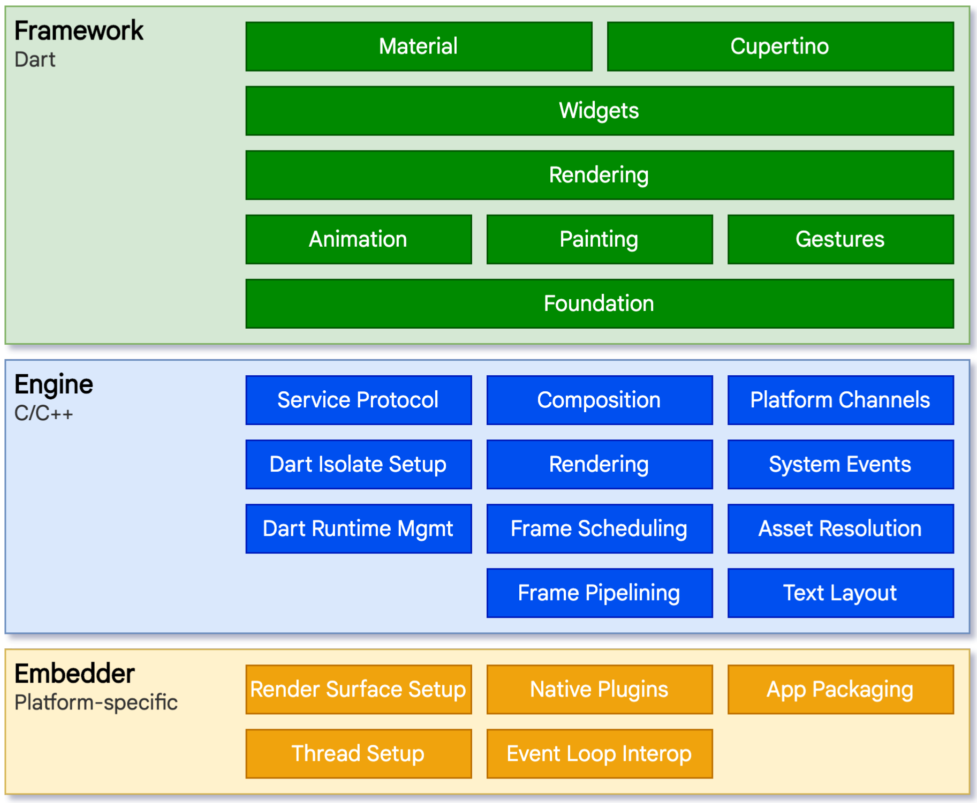

Figure 2.1: Flutters architectural layers[8]

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 graphics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compiles toolchain. It is responsible for rasterizing visuals when

[8]Flutter architectural overview, `https://flutter.dev/docs/resources/architectural-overview`, Accessed 2021-02-08
[9]Ibid.
[10]Ibid.

a new frame needs to be painted.[11]

Framework

Developers will usually interact with Flutter using the Framework, which is composed of a series of layers. The bottom layers are the foundation classes. Classes for building block services like animation, paintings, and gestures that use common abstractions over the underlying foundation.[12]

Above that is the rendering layer. It provides an abstraction for dealing with the layout. It consists of a tree of renderable objects. These objects can be manipulated dynamically. If there are any changes, the tree will automatically update the layout to reflect the changes.[13]

The widgets layer is an abstraction of the rendering layer. Each rendered object has a corresponding class in the widgets layer. The Material and Cupertino are examples of two packages using the widget layers components. Components from the Material package implements the Material design language[14], and respectively, components from the Cupertino package are implementing the iOS design language[15]. The philosophy is the widgets should produce appropriate effects of the platform conventions but not adapt automatically when app design choices are needed[16].[17]

The Flutter framework also implements packages including higher-level features that a developer might need. Packages supporting features camera and webview, as well as platform-agnostic features like characters, HTTP, and animations.[18]

[11]Ibid.

[12]Flutter architectural overview, `https://flutter.dev/docs/resources/architectural-overview`, Accessed 2021-02-08

[13]Ibid.

[14]Material Design - `https://material.io/`

[15]Human Interface Guidelines, `https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/`, Accessed 2021-02-08

[16]Platform specific behaviors and adaptations `https://flutter.dev/docs/resources/platform-adaptations`, Accessed 2021-02-14

[17]Flutter architectural overview, `https://flutter.dev/docs/resources/architectural-overview`, Accessed 2021-02-08

[18]Ibid.

Widgets

The framework and applications UI main building blocks are widgets. The idea is to use widgets when developing UI for Flutter applications[19]. Widgets can either be stateless or stateful. Stateless widgets are immutable, where stateful widget are always mutable[20][21][22].

A widgets state is store in a *State* object. The state consists of the mutable values of the widget[23]. If the widgets state changes, the widget is responsible for notifying the state object that it has changed[24]. Thus, when the widget changes, the framework will be notified and rebuild the UI. See more in 2.1.1.

Rendering

The Dart code that paints Flutter's graphics compiles into native code. It uses the open-source 2D graphics library Skia[25]. It does not use an abstraction layer that might use underlying native libraries. It bypasses the platform's system UI libraries and uses its own widget set. It enables Flutter to skip adding an overhead that could be significant, which might affect an application's performance. Particularly if there would be several interactions between the UI and the application logic.[26]

Flutter embeds a copy of Skia as part of the Flutter engine. It lets developers update their applications and stay up to date with the latest improvements. Even if the versions of the platforms do not update. The same goes for Android, iOS, macOS, or Windows.[27]

[19]Introduction to widgets, `https://flutter.dev/docs/development/ui/widgets-intro`, Accessed 2021-02-08

[20]StatefulWidget class, `https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html`, Accessed 2021-02-08

[21]StatelessWidget class, `https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html`, Accessed 2021-02-08

[22]Adding interactivity to your Flutter app, `https://flutter.dev/docs/development/ui/interactive`, Accessed 2021-02-08

[23]Ibid

[24]StatefulWidget class, `https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html`, Accessed 2021-02-08

[25]Skia Graphics Library - `https://skia.org/`

[26]Flutter architectural overview, `https://flutter.dev/docs/resources/architectural-overview`, Accessed 2021-02-08

[27]Ibid.

Platform embedding

To the underlying operating system of the platform, Flutter applications are packaged in the same way as native applications. The embedders are platform-specific and are written in the language appropriate to the platform. For Android, the embedder is built using Java and C++. For iOS, the embedder is built using Objective-C/Objective-C++.[28]

The platform embedder is the native application that hosts the Flutter content. Its binds together the host operating system and Flutter. When starting a Flutter app, the platform embedder initializes the Flutter engine, obtains threads for the UI and rastering, and creates a texture which Flutter can write to. The embedder is responsible for the application lifecycle, input gestures (mouse, keyboard, touch, etc), window sizing, thread management, and platform messages. Flutter currently include embedders for Android, iOS, Linux, macOS, and Windows.[29]

On Android, Flutter loads in the embedder as an *Activity*. The view is begin controlled by a *FlutterView*. The *FlutterView* renders Flutter content as a view or a texture, depending on the composition and z-ordering of the Flutter content.[30]

On iOS, Flutter is loaded in the embedder as a *UIViewController*. The platform embedder creates a *FlutterEngine* which is the host to Dart VM, Flutter runtime and a *FlutterViewController*. The *FlutterViewController* attaches the *FlutterEngine* to pass UIKit or CocoaInput events into Flutter and display frames rendered by the *FlutterEngine* using Metal or OpenGL.[31]

2.2 Native mobile applications

Native mobile applications are developed to be able to run on a specific platform, such as Android or iOS. Native applications cannot be executed from other platforms. Platform vendors often provide developers with an Integrated Development Environments (IDE), native programming language, APIs, and application distribution market [15] and Software Development Kit (SDK).

[28]Ibid.
[29]Flutter architectural overview, `https://flutter.dev/docs/resources/architectural-overview`, Accessed 2021-02-08
[30]Ibid.
[31]Ibid.

They often have the benefit of taking advantage of the device's various features, such as a camera and sensors.

2.2.1 Development

Each platform usually has its development process with its native programming languages. Android applications can be developed by using Kotlin, Java or C++ [27, 28], while iOS applications can be developed by using Swift or Objective-C [29].

Developing a native application is often developed with the SDK provided by the vendor. The SDKs include development tools and common user interface elements. Interface elements as buttons, text, navigation components, etc. [27, 29]

This means that the development of native applications is tightly coupled with its platform. This prevents developers to develop one application for multiple platforms where the code is reused. If an application needs to support multiple platforms, it is necessary to have multiple codebases.

2.3 Model-View-ViewModel

According to Sorensen and Mikailesc [30], 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 fig. 2.2). [30, 31]

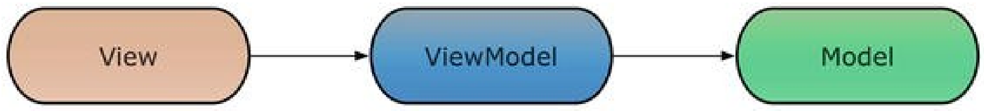

Figure 2.2: The *Model-View-ViewModel* (MVVM) design pattern [31]

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. [31]

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 [32, 33]. In research by Lou [32], 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.

Aljamea and Alkandari [33] investigates both MVC and MVVM for iOS development. The authors claim that using the MVC pattern tends to have massive files containing the code for both the view and the controller. Which makes it challenging to test, reuse, and maintain. Additionally, it increases the risk of writing errors in the code without recognizing them as they are long and difficult to test. Therefore, the authors conclude their research by advising developers to use the MVVM pattern as it separates concerns making it more testable and modifiable.

2.4 t-Test

According to Quirk [34], two groups are independent of each other whenever the groups consist of different people. Conditionally, no person can be in both groups and every person only produces one value in number in statistical measurements. When studying the independent groups, it is often required to statistically test central tendencies (mean or median) of the two groups, whether they are significantly different from each other or not[35].

In 2006, Ruxton [35] evaluated 130 research papers and found that there were mainly three different tests used to find significant changes between two groups. The three common tests were: Student's t-test, Mann–Whitney U test, and the t-test for unequal variances. In the 130 papers evaluated the majority (47 out of 71) of them had a sample size below 30 participants.

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 [35] 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:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{2.1}$$

where $\overline{X_j}$, s_j and N_j are the jth sample mean, sample standard deviation, and sample size, respectively $j \in \{1, 2\}$. If the t-value is below 0.05 there is a significant difference between the group with a 95% confidence[36].

2.5 User experience questionnaire

For evaluating interactive products, user studies can be conducted to draw qualitative and/or quantitative conclusions. With the help of feedback from individuals participating in a study, conclusions can be drawn. Users participating in the study need to be the target group of the interactive product. Otherwise, incorrect conclusions could be made.[37]

One of these studies could be made using a User Experience Questionnaire (UEQ). It was created by Schrepp [37] in 2005 using a data analytical approach to create a practical relevance of constructed scale items. In other words, the scales were derived from data from bigger sets of scale items. The results were six scales that describe different aspects of the user experience.

UEQ scales

- Attractiveness - *Overall impression of the product. Do users like or dislike the product?*
- Perspicuity - *Is it easy to get familiar with the product? Is it easy to learn how to use the product?*
- Efficiency - *Can users solve their tasks without unnecessary effort?*
- Dependability - *Does the user feel in control of the interaction?*
- Stimulation - *Is it exciting and motivating to use the product?*
- Novelty- *Is the product innovative and creative? Does the product catch the interest of users?*

UEQ consists of multiple scale items. Thus, each scale item is a scale question about the product which the participant should answer or leave blank. The questions are created by the creators of the UEQ and are thus predefined. The scale items are not replaceable with custom questions. The UEQ and the scale items are available in multiple languages at UEQ's official website[32]. The scale item ranges from a value of -3 to 3. The middle value is 0, which is considered neutral. One item regarding the attractiveness of a product looks like this:

attractive ○ ○ ○ ○ ○ ○ ○ unattractive

Participants in the study evaluating the product should answer the UEQ immediately after using the product. If participants fill out the UEQ after discussions about the product, it may influence the results. The goal of the UEQ is to catch an immediate impression of a user towards the product. Thus, discussion about the product may come afterward. [37]

More data collected using the UEQ will give a more stable scale means, and thus more accurate conclusions can be drawn from the data. There is no defined minimum number to use the UEQ. However, according to Schrepp [37], products evaluated go about providing reliant results from 20-30 participants. The reliability of data also depends on the participant's level of agreement. The more the participants agree, the lower the standard deviation of the answers to items is, and less data is needed for reliable results.

The data gathered from the UEQ can be analyzed using an Excel-tool. The tool is provided by the creators of UEQ and is available at their website[33]. It aims

[32]User Experience Questionnaire - `https://www.ueq-online.org/`
[33]User Experience Questionnaire - `https://www.ueq-online.org/`

to make the analysis of the UEQ data as easy as possible. They also provide a tool designed to be used when comparing two different products that have been evaluated using two different UEQ's.

The comparison tool produces a graph that can be used to control if the results have a significant difference. It calculates the mean values of the UEQ scales with a 95% confidence interval. The confidence interval is represented by error-bars in the graph.

Assuming it would be possible to repeat the evaluation indefinitely under the same conditions, there would some random influences which would affect the results. However, with a 95% confidence interval, it is the interval in which 95% of the scale mean repetitions should be located hypothetically. Thus, the confidence interval shows how accurate the measurements are. If the confidence interval is relatively large, it indicates that the measurements are inaccurate.

The comparison tool also conducts a two-sample t-test. It uses a selected α value of 0.05 which indicates how statistically different the results are. If the t-test value is below the α value 0.05, there is a significant difference between the two versions. This can be seen in the graph produced by the tool, using the error-bars. If the error-bars overlap, there is no significant difference among the versions.

Chapter 3

Method

This chapter presents the methodology of the study. Beginning by describing the sample application which was developed. Followed by describing how a look and feel user study was conducted. Lastly, the chapter finishes by describing the performance metrics measured and how were measured.

3.1 Developing sample applications

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 (see 1.3).

3.1.1 Complexity

In previous studies (see. 1.6.1) 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 and therefore no internet connection was needed. Both of their sample applications were dependent on using mock data.

Sample applications developed by Evert [17] and Hansson and Vidhall [12] needed the internet as they communicated with different APIs. Evert [17] application used an external REST-API for searching users on GitHub and view their public repositories using two different endpoints. Hansson and Vidhall [12] developed a home automation application that also used a REST-API, although using a private internal API. Both Evert [17] and Hansson and Vidhall [12] 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 [17] and Fredrikson [18] sample applications.

As mentioned in section 1.6.1, 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. Thus, answer the research questions (see. 1.3).

3.1.2 Design pattern

The most common and used design patterns for Android and iOS development are Model-View-Controller (MVC), Model-View-Presenter (MVP), and Model-View-ViewModel(MVVM) (see more in 2.3). According to previous research, the MVVM design pattern seems to be more preferably regarding modifiability, testability, and performance. The research indicates MVVM might be the better alternative. Thus, valid for developing production applications. Therefore, the aim was to use the MVVM pattern when developing the sample applications for this research and thesis.

3.1.3 User experience and user interface

Prior studies have shown that cross-platform applications tend to have inferior UI/UX than native applications (see 1.6.2). However, one study concluded that it was possible to achieve a similar user experience using the cross-platform technology React Native[12]. Therefore, this thesis aimed to evaluate if the same goes for using the UI toolkit Flutter. This meant, when developing the cross-platform application, the intention was to achieve an application that had an identical look and feel like a native application. This was accomplished by not including any complex functionalities which might be challenging to develop using the toolkit. For example, any augmented reality functionality would be challenging include, as the toolkit does not have any documentation or official support for AR functionalities regardless that both Android and iOS might have AR SDKs for their respective platform.

Additionally, a third-party library name *flutter_platform_widgets*[1] were incorporated when developing cross-platform sample application using the UI toolkit Flutter. This ensured the cross-platform sample application graphical components would be similar to the native components that might have been available from the native SDKs. When using the widgets available through the library, it renders *Material* widgets for Android and the *Cupertino* widgets for iOS.

3.1.4 Developed applications

With the considerations regarding complexity and UI/UX, the chosen sample application for this thesis was a movie finding application. All of the necessary data were retrieved using The Movie Database API (TMDb API)[2]. By sending an HTTP request to the API, data was retrieved by parsing JSON data from the HTTP response. The sample application had three major functionalities:

1. Show currently trending movies.
2. Search for a movie.

[1]flutter_platform_widgets `https://pub.dev/packages/flutter_platform_widgets`

[2]The Movie Database `https://www.themoviedb.org`

3. Show details about a movie.

The first functionality was implemented by presenting the user with a list of trending movies. If the user scrolled down to the end of the list, more trending movies were loaded. This worked by sending an HTTP request to the TMDb API which resulted in an HTTP response with JSON data of the currently trending movies. The API also supported pagination, thus the functionality of loading more movies while scrolling was made by making new HTTP requests when necessary.

The second functionality was implemented by presenting the user a search text field. When the user searched for a movie, an HTTP request was sent to the TMDb API. If the API resulted in any search hits, the JSON data was retrieved and parsed from the HTTP response and the user has presented a list of search hits.

The third functionality was an added functionality dependent on either of the first two functionalities. When selecting a movie from the trending list or the search results, the user was presented a view with more details about the selected movie. Necessary movie detail information was available from previous functionality. Thus, necessary movie details were available from both endpoints which were used when loading trending movies or searching for a movie. Furthermore, the movie's poster was presented with the movie details. The poster was loaded asynchronously after populating the movie details view. This was possible due to the path to the image was part of the API response from previous functionality with the movie details. With the three functionalities combined, an overview of the developed applications' different states can be seen in the figure 3.1 below.

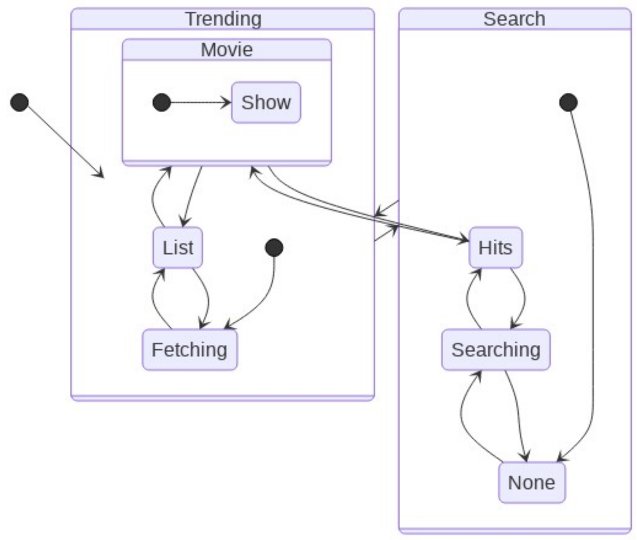

Figure 3.1: State diagram of the developed sample applications

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.

Detailed explanation of each state

- **Trending**
  - **Fetching:** Fetches trending movies immediately after startup using TMDB API with endpoint *api.themoviedb.org/3/trending/movie/day?page=<PAGE>*. Fetches additional movies when user reach ends of list (pagination). Movies already loaded are still visible to the user.
  - **List:** Loaded trending movie list visible to the user.
- **Search**
  - **Searching:** Searching for desired movie using

TMDB API with endpoint
api.themoviedb.org/3/search/movie?query=<QUERY>

  - **Hits:** Search results visible to the user.
  - **None:** No movies found by searching or the search text field is empty. If the search text field is empty user is not searching for anything.

- **Movie**
  - **Show:** Selected movie visible to the user.

    Movie poster image loaded asynchronously from
    image.tmdb.org/t/p/w342/<MOVIE POSTER PATH>
    without changing states.

The three core functionalities implemented, are fundamental functionalities used that are used by several real-world applications. Internet communication, pagination, and parsing of JSON data. Additionally, a movie was considered sufficient complex data object for the business logic.

In summary, the sample applications developed had functionality similar to a real-world application. Furthermore, the cross-platform developed applications aim to mimic the characteristic of a native application's user experience.

MVVM and Riverpod

The MVVM design pattern was used when developing native applications. Both for the native Android application as well as the native iOS application. However, no valuable or significant resources were found regarding developing Flutter applications using the MVVM pattern. Therefore, the alternative design pattern Riverpod[3] was used.

Riverpod is one of Google's many recommendations regarding design patterns[4]. It was also understood to function similarly to the MVVM design pattern. As in MVVM, Riverpod has a component named Provider which glues the View and Model components together. Additionally, the view holds a reference to the Provider, while the Provider holds a reference to the Model as in the MVVM pattern. Thus, similar to the figure 2.2.

[3]Riverpod `https://riverpod.dev/`

[4]List of state management approaches `https://flutter.dev/docs/development/data-and-backend/state-mgmt/options`

3.2 Performance metrics

One of the focal points of this thesis is to study the user-perceived performance of a Flutter developed application. Therefore the choice was made to measure application size and startup time.

According to Willocx, Vossaert, and Naessens [38], application size is mainly important for low-end devices. Low-end devices tend to have limited persistent memory available. Furthermore, the installer size of an application is also significant for those who download an application using mobile Internet [39]. This suggests that both installer and installed application size are key for reaching out to a bigger audience.

Prior study has shown that cross-platform technology tends to produce an application larger than a native application[38]. Thus, it may be interesting to investigate if the same applies to a Flutter developed application.

As noted by Singh, Agrawal, and Kanukotla [40] startup time is one of the most vital performance metrics for user experience. Due to the usage of several applications, a significant amount of memory is consumed resulting in decreased phone performance. This is visible when launching an application. Thus, the author of this thesis believes that measuring application startup time may be a key performance metric. It is a crucial metric for user experience, hence, important from a user's perspective.

A prior study by Willocx, Vossaert, and Naessens [38] found that cross-platform technology based on JavaScript frameworks tends to take a longer time to startup than a native application. The overhead is largely caused by loading a web-view, which is the foundation of the frameworks. Thus, it is of interest to investigate if the same applies to a Flutter developed application.

3.2.1 Size

Measurement of the installer was primarily done by measuring the size of the installer file. Thus, for Android installer .aab, and likewise for the iOS installer .ipa the size of the file was measured.

For the measurement of the application size and how much disk space was needed for the installed sample applications, the built-in platform features were used. On both Android and iOS, the application size could be found under the settings menu. This information was also available to the user. Thus, achieving a user-perceived application size.

3.2.2 Startup time

The startup time of the sample applications was measured using Firebase Performance Monitoring[5]. It was incorporated by including the Firebase Performance library to each of the different codebases of the developed sample applications. For both Android and iOS, the tool automatically collects duration traces related to the application lifecycle. A duration trace is essentially a timer which captures data between two points in time in the application. The startup time is measured using a app start trace. The duration trace measures the time between when the user opens the app and the app is responsive.

For Android, the app start trace starts when the application's *FirebasePerfProvider ContentProvider* completes its *onCreate()* method. It stops when the first activity's *onResume()* method is called.

For iOS, the app start trace starts when the application loads the first *Object* to memory. It stops after the first successful run loop that occurs after the application receives the *UIApplicationDidBecomeActiveNotification* notification.

To gather real user data, the startup time were measured every time the applications were by an user. This was possible due to a user perception study conducted on the sample applications. The participants had installed one sample application on their mobile device and then performed necessary tasks for the study. When the applications were launched, the startup time was measured by the monitoring tool. This ensured that the data was qualitative and reflected real-world application startup time.

3.3 User perception study

To investigate the user perception between Flutter and native applications, a user study was conducted. The goal of the study was to have quantitative data to evaluate if the users showed any signs of preferring Flutter developed applications over native applications or vice versa. A quantitative approach was chosen to obtain a general understanding of the user perspective.

3.3.1 Why UEQ?

The UEQ method was chosen as it produces reliable results in a relatively small amount of participants. It was observed that 20-30 participants often tend to

[5]Firebase Performance Monitoring - `https://firebase.google.com/products/performance`

give reliable results (see more in 2.5) for one product. Thus, it would need 80-120 participants as there are in total four applications being tested. However, in the study by Hansson and Vidhall [12] which used the same method for evaluating the cross-platform framework React Native had conclusive results from 32 participants testing four different applications. Fredrikson [18] conducted a similar study using the same UEQ method and found conclusive results with 30 participants testing three different applications.

The UEQ also provided comparison-tools calculating statistically significant differences between two different products. It calculated the statistical difference using a t-test with unequal variances. The t-test with unequal variance has been recommended approach by Ruxton [35] (see 2.4) since 2006. Using t-test with unequal variances is more likely to produce reliable results when evaluating differences among two independent groups than other similar methods. In the same study, it was found the majority of studies related to evaluating significant differences in two independent groups, had below 30 participants in the tests.

Based on the findings of Ruxton [35] and previous studies which used the same method by Fredrikson [18] and Hansson and Vidhall [12], the aim was to get at around 20-30 participants for each comparison. Resulting in a total of 40-60 participants. Additionally, applying the t-test using unequal variances on the results of the UEQ would indicate if there were any significant statistical differences between the different applications.

3.3.2 Conducting the UEQ

The study was initiated by asking the users to perform two different tasks on the sample application. After completing both tasks, the user was asked to immediately answer the UEQ. The questionnaire used is available in Appendix A. After the users had answered the questionnaire, the results were run through the UEQ data analysis tool. It generated mean values and with a confidence interval.

To execute the study, the sample applications were made available remotely on Android and iOS respective application distribution platforms as a beta application. The participant could then download the applications on their own devices and participate in the study in their most convenient way. A participant who owns an Android device could either test the Flutter developed application or a native Android application. Respectively for participants owning an iOS device, could either test the Flutter developed application or the native iOS application. Participants were not made aware of which version of the sample application they tested.

For convenience, and to encourage more individuals to participate in the study, face-to-face tests were performed. Meaning, an Android or iOS device was handed to a user, which they then performed the tasks on. Immediately after the tasks were performed, they were asked to answer the UEQ. Similar to the participants who tested the applications remotely, these participants were not aware of which version of the sample application they tested. The face-to-face test was conducted without giving them any additional information that would not be available for the participants who tested the applications remotely on their devices.

User tasks

The users' tasks described below, were chosen to ensure the participants in both groups tested the application at a minimum. No matter how much time each participant spent testing the app, participants from both groups had tested all of the functionalities of the application.

1. View details about any trending movie.
2. Search for any movie. If search results
   (a) includes desired movie $\rightarrow$ view details about the movie.
   (b) does not include desired movie $\rightarrow$ search for another movie and repeat the task until desired movie is found.

Dataset

When gathering data for the user perception study, the participants were first asked what their default platform was. If it was Android, they tested a sample application for the Android platform. Respectively for iOS users, they tested one of the iOS sample applications.

This resulted in 23 Android users and 23 iOS users. In a total of 46 participants in the study. Of the Android users, 11 participants tested the Flutter Android application and 12 participants tested the native Android application. Of the iOS users, 11 participants tested the Flutter iOS application and 12 participants tested the native iOS application.

There were no specific target group of participants. The only requirement was that they were using an Android or iOS device regularly. This was done to decrease the threshold for acquiring participants to the study. The sample applications could also be used by anyone.

This lead to a group of individuals with different backgrounds, different ages, and different technical knowledge. The application was deemed to have such limited functionality, that none of those factors would have resulted in any significant effects on the study.

Processing the data

After the execution of the UEQ, the data was inserted in the provided UEQ comparison tool (see more in 2.5). It processed the data and computed the necessary table and graphs needed for this thesis. Furthermore, t-tests using unequal variances were conducted to find the statistically significant differences. See more about the t-test in section 2.4.

The UEQ does allow null values, meaning the participants could choose to not answer a question. Although, all questions had been answered by every participant.

Chapter 4

Result

This chapter presents the results of the research conducted for the thesis. It starts by presenting the results regarding the user-perceived performance. There is one section each for application size and startup time. It is followed by presenting the results of the user perception study conducted using the UEQ method.

4.1 User perceived performance

4.1.1 Size

On both Android and iOS, the installation package size and the installed application size were larger than the native applications package size and installed application size. This is due to the overhead of the Flutter engine and framework. Even if an application is quite minimal, such as the sample applications, the Flutter core is still necessary. Hence, the difference in size between Flutter 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 approximately 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

Table 4.1: 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 4.2.

	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

Table 4.2: 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.

4.1.2 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.

Android

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

Table 4.3: Startup time of the developed Android sample applications.

iOS

The native iOS application was tested on five different devices, and the Flutter application was tested on another four devices. In a total of nine different devices. However, there were only three different models. The Flutter developed 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

Table 4.4: Startup time of the developed iOS sample applications.

4.2 User perception study

The user perception study showed that there were no significant differences between the developed sample applications. No significant differences between the Flutter applications and the native application for either the Android or iOS platform. Indicating that users do not have any preferences between Flutter applications and native applications, as no significant differences could be found.

4.2.1 Android

The values of the UEQ scale means for the developed Android sample applications can be seen in figure 4.1. The native Android application scored higher on most UEQ scales. However, there are no statistical differences that can be seen in the results of the t-test with unequal variances in figure 4.5. Furthermore, in figure 4.1 all of the error bars overlap with each other, which further indicates there are no significant differences between the two applications.

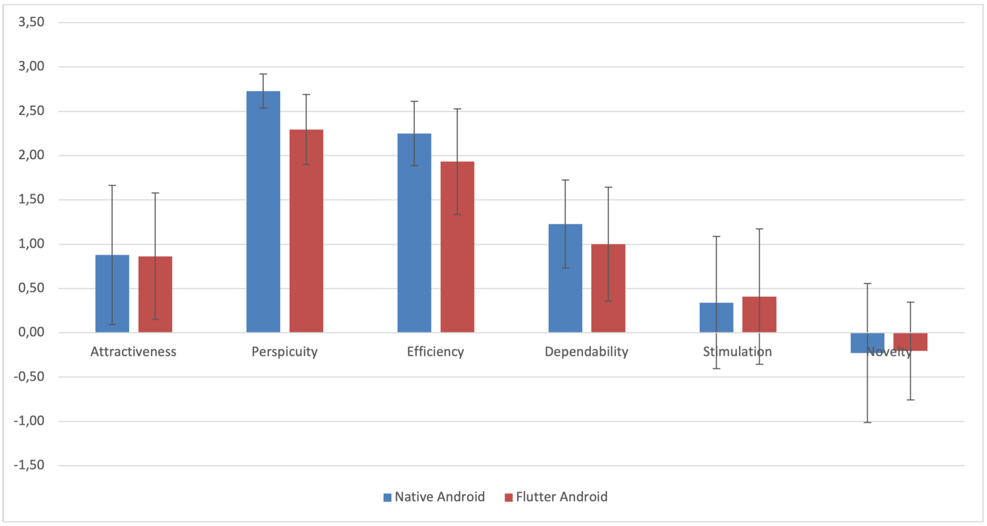

Figure 4.1: Comparison of UEQ scale means for the Android applications.

UEQ Scale	t-Test Value	Significant Difference
Attractiveness	0.9780	No
Perspicuity	0.0741	No
Efficiency	0.3834	No
Dependability	0.5903	No
Stimulation	0.9017	No
Novelty	0.9635	No

Table 4.5: Two sample t-test assuming unequal variances results on the UEQ mean values for the Android sample applications.

4.2.2 iOS

The values of the UEQ scale means for the developed iOS sample applications can be seen in figure 4.2. The Flutter application scored higher on all UEQ scales. Similar to the case for the Android platform, there is no statistical difference between the two iOS applications. Results of the t-test with unequal variances show (see table 4.6) shows no significant differences and all of the error bars overlap with each other.

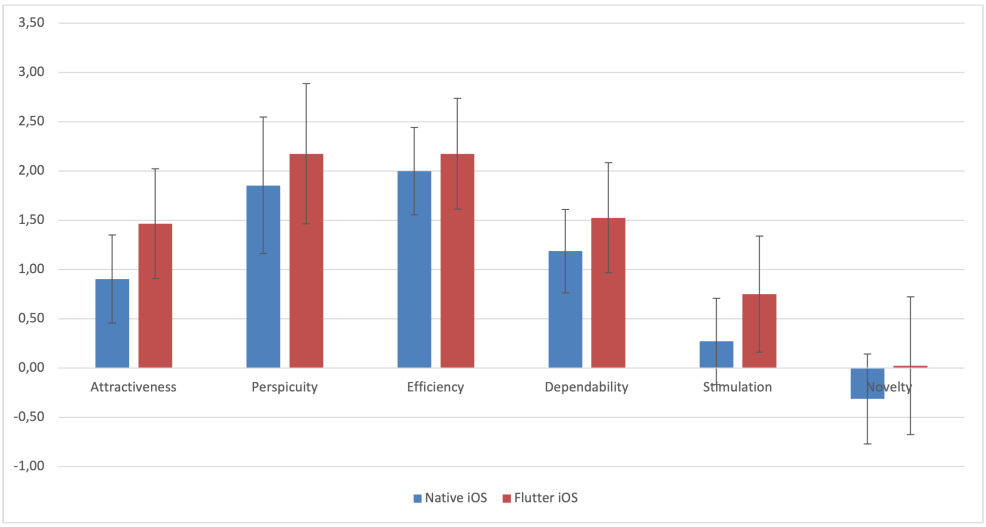

Figure 4.2: Comparison of UEQ scale means for the iOS applications.

UEQ Scale	t-Test Value	Significant Difference
Attractiveness	0.1389	No
Perspicuity	0.5345	No
Efficiency	0.6373	No
Dependability	0.3578	No
Stimulation	0.2171	No
Novelty	0.4394	No

Table 4.6: Two sample t-test assuming unequal variances results on the UEQ mean values for the iOS sample applications.

Chapter 5

Discussion

This chapter presents the discussion of the results and findings during the research for the thesis. It starts by discussing the measurements and results of the user-perceived performance research. Followed by the discussion and analysis of the user perception study. Lastly, there are sections regarding the source of error, sustainability, and ethical aspects

5.1 User perceived performance

The first question of this thesis was to find what is the effect on user-perceived performance by a Flutter developed application? The results of this thesis indicate that the user-perceived performance is inferior compared to the user-perceived performance of native applications. For both the Android and the iOS platforms. It was found to be consistently inferior in both of the investigated user-performance metrics: application size and startup time. A possible explanation of this might be due to the overhead of the Flutter engine.

5.1.1 Flutter overhead

The core of Flutter is the Flutter engine. It acts as the conductor and directs how a Flutter application behaves. The engine is similar to a game engine, as the engine decides what, when, and how the graphics should be rendered. The engine is responsible for rasterizing visuals when a new frame is needed. Additionally, the Flutter engine bypasses the platform's UI library to render its UI components with its own widget set. This means there is no way to bypass or reduce this overhead. It needs to be embedded in all Flutter applications and is the most essential part of a Flutter application.

This results in the Flutter engine being an overhead to every Flutter application. Presumably, this overhead is an initial cost as the overhead is constant. It is less of an issue when applications grow in terms of complexity or application size. As the overhead is constant, the ratio of the overhead will decrease as application size and complexity grows. There is no correlation between Flutter's overhead with application size and complexity. It is only a necessary fundamental building block.

5.1.2 Size

This thesis found that for both Android and iOS platforms, the developed Flutter application was significantly larger than its native counterpart. It is assumed this is due to necessary Flutter overhead. Flutter bypasses the platform's UI library and has its widget system, therefore it needs to embed the widget system

No official documentation was found if Flutter contains all of the widgets or only the widgets necessary for a developed Flutter application. There is a risk that redundant widgets are added. This could be one pitfall of why the Flutter applications were significantly larger. However, it is certain that Flutter includes a copy of the Skia graphics library (see section 2.1.1) and thereby increases the application size.

Having an embedded widget system and include a copy of Skia might be detrimental to the application size. It does have its benefits as well. Flutter does not need to have an abstraction layer that similar cross-platform technologies (see section 2.1.1) would need instead.

Furthermore, this allows Flutter to be developed without adapting to changes for each platform. It might be beneficial for developers as well as the maintainers of the Flutter. When any platform updates its versions and its native UI libraries, it would not affect existing Flutter applications. It is beneficial for developers as they only need to adapt to changes in Flutter, and not track changes for each platform. For the maintainers of Flutter, it results in not having to adapt to every new release of platform versions. Which likely implies that there is less time needed to make Flutter work and more time to improve the existing technology. Where maintainers of similar technologies might need to spend time adapting their solution for each new version of every supported platform. It is necessary for them as their solution might use an abstraction layer and thus be dependent on the native UI libraries.

The results of having the opposite approach would presumably yield better results in terms of application size. By not using its widget system, use native UI libraries, and not embed a copy of another graphics library would decrease the application size. This can be seen in other research evaluating similar cross-platform technologies. Kotlin Multiplatform and React Native cross-platform applications showed a maximum of 18% larger Kotlin Multiplatform apps and respectively 5-15 MB larger React Native application than their native counterparts (see 1.6.3). The Flutter-developed sample applications were seven times the size of a native Android application and nine times the size of a native iOS application.

If the application size is an important factor for an application, a Flutter application is not a viable alternative for a native application from a user-centered perspective. Even if the overhead of the Flutter application is consistent, the overhead may be significant enough for users not to use the application. Using another cross-platform solution such as Kotlin Multiplatform or React Native should result in a lower application size.

5.1.3 Startup time

In the current study, comparing the Flutter application's startup time with the native application's startup time showed that the mean startup time was lower for the native applications. For both the Android and the iOS platforms. A qualitative assumption is that it is due to Flutter's internal overhead.

Flutter applications are dependent on the core engine, which in turn is dependent on the graphics library Skia. This implies when launching a Flutter application the engine must first be initialized and in turn graphics library also needs to be initialized. This is most likely the cause of why Flutter applications seem to launch slower than their native counterparts.

The startup time was measured on different device and models which have different hardware, and thus difference performance. The correlation between startup time and hardware or operating system was not investigated. However, one can assume newer devices probably have better hardware than older devices, which results in better performance. The results also seem to indicate this. It is important to bear in mind that that is not might be the case. There is not enough empirical data to validate this idea.

One idea that can be considered, is that Flutter developed applications seems to take longer time to start than native applications and similar cross-platform solutions. In prior studies, the startup time is quite similar between native and cross-platform developed applications [17]. A note of caution is due here since there ware significant difference between lower and upper bounds of the startup time. To increase the confidence of the conclusion, the startup time would need to be measured on more devices and thus require more participants in the user study.

Another major source of uncertainty is in the method used to measure the startup time. It does not consider the difference between cold and hot startup times. Cold startup time would the the measurement of the time taken when the application started from scratch, where the system's process has not been previously started. Hot startup time would be the measurement of the time taken where the system's process had already been started. However, if there would be a significant number of users tested the application these difference would likely not affect the end result. If there was vast amount of quantitative startup time data available, it would be sufficient to draw conclusion based on the median measured by Firebase Performance Monitoring. The small size of the dataset indicates the tool was not appropriate chose for this research. In conclusions, it can thus be suggested that Flutter native application seems to take longer time to startup than native applications but the uncertainty is high.

5.2 User perception study

The most obvious finding to emerge from the user perception study is that there are no significant differences between the developed Flutter application and the developed native applications. For both platforms, participants in the study did not perceive any of the applications significantly differently.

A note of caution is due since the confidence intervals are relatively large on most of the UEQ scales. Indicating the measurements might be inaccurate and insufficient. This is likely due to the small sample size in the study. According to the creator of the UEQ method, 20-30 participants for each product tend to give reliable results. The user perception study for this thesis had around ten participants for each product. Thus, any conclusions regarding based on the user perception study will be uncertain.

However, it is doubtful more participants would change the outcome of this study. Even if the confidence interval is relatively large, there is an overlap for all of the UEQ scales for both the Android and iOS platforms. The t-test with unequal variances further statistically validates that there are no significant differences between the developed applications. Implying, there is no preference from the users between native and Flutter applications.

These findings suggest that it is possible to achieve the same look and feel as native applications. This may be explained by the fact the Flutter's adaption philosophy is to produce the appropriate effect of the platform conventions but not adapt atomically to new changes in the platform (see section 2.1.1). I.e. Flutter aims to mimic native application conventions. It can thus be suggested that the results of the conducted user perception study are only valid as long as the adaption philosophy is consistent and up to date with the platform conventions. If the platform conventions changes due to an updated version and Flutter does not get updated to adapt to the newer conventions, there would be a discrepancy between Flutter developed applications and native applications. Consequently, this would affect users as they might get accustomed to and expected a certain behavior while using their device. Thus, perceive Flutter application differently for better or worse.

5.2.1 Method

It was harder than expected to convince people to participate in the user perception study. For many individuals, downloading and installing an application on their device, test it, answer a questionnaire was too much of a hurdle. Hence, the method might have been the wrong one.

All of this also had to be done remotely due to the COVID-19 pandemic in Sweden, which entailed that it was difficult to understand the participants. Did they understand the instructions, did they understand the application, or did they need more initial guidance? Also, did this affect their perception? The UEQ is still most likely the right tool for the job, it should probably not have been conducted remotely.

5.3 Sample applications

To the best of the author's knowledge, there does not exist identical applications developed in Flutter, natively in Android, and natively in iOS which are available for research purposes. Therefore, the method was chosen to develop sample applications with identical features and functionality.

The features chosen for the sample applications are features that are present in production applications. Features such as searching for items, displaying data in a scrollable list, and view further details about an item. The functionalities chosen for the sample applications are also functionalities that are present in production applications. Fetching data from an external API, parsing JSON data, and using an endpoint that uses pagination. This was deemed to mimic a production application sufficiently, for concluding whether a Flutter developed application is a viable alternative for a native application from the user's perspective.

The aim was to reduce disparities between the sample application and production applications, which does not mean there do not exist disparities between the sample applications and production applications. Consequently, it does affect the results of the thesis. If different applications with other features and functionalities had been chosen, the results might have been different. It could affect the application size, startup time, and the user's perception.

However, the outcome or findings of the user-perceived performance would most likely be the same. One of the main findings and assumed biggest significant factor for inferior user-perceived performance is the built-in overhead of the Flutter engine. The overhead of the Flutter engine is the fundamental block that is constant. Thus, no matter what features and functionalities are chosen when developing a Flutter application, it would be the same. The same issues and difficulties arising from Flutter's inner workings would arise for any Flutter developed application.

Similarly, it would doubtfully change the outcome and findings from the user perception study. Conditionally, that the applications are limited to used relative limited features and functionalities as in this study.

If the same research would have been made on applications that were dependent on functionalities only available through native libraries, it would most likely affect the user perception. Whether in favor or disfavor of the Flutter application can not be said until it has been tested. Furthermore, no features were deemed to be resource-intensive, which implies it difficult to say what the effects would be on user perception if resource-intensive features were necessary.

5.4 User's perspective

The present thesis was designed to determine: can a Flutter developed application be a viable alternative for native application from the user's perspective? Concerning the findings of the user perception study, the answer would be yes.

This conclusion might seem like a contradiction to the evaluation of the user's perceived performance. Flutter application has an inferior performance. However, it does not seem to affect the user's perception of the application. It can thus be suggested that Flutter overhead, which causes the performance issues is not significant enough to affect user perception of the application. Even if a Flutter application might take a little longer time starting, it does not affect how the user perceives the application.

However, a source of uncertainty is the correlation between the application size and user perception. The application size is not visible to the user when the application launches or when an application is being used. Thus, it is uncertain if the application size would affect the user perception of the application. Thus, if application size is an important factor, then a Flutter developed application might not be a viable alternative.

It is known that application size matters more in emerging markets. This implies the importance of the application size differs in different use cases and markets. If the aim is to develop applications for emerging markets, Flutter should not be the chosen alternative. The performance cost of using Flutter with its overhead in application size might discourage the user to use a Flutter developed application. It might even inhibit individuals' access to the applications as users in emerging markets might have limited persistent storage.

5.5 Design pattern

There were two different design patterns used when developing the sample applications. Model-View-ViewModel (MVVM) was used for the native applications, while Riverpod was used for developing the Flutter applications.

Even though one study indicates design patterns may affect results regarding the performance, it is still doubtful that it does have any significant effect on performance. Studies related to design patterns mostly focus on testability, modifiability, and similar aspects regarding the written code. There was only one study found related to performance and design patterns in mobile application development. The study investigated consumed memory and not user-perceived performance. Thus, not affecting the results of this thesis.

Using the same design patterns consistently on all the developed applications would doubtfully change the outcome of the user perceptions study. The view components would be the same and thus the UI/UX would be the same.

In previous related work where the authors also developed sample applications for evaluating performance, UI/UX, and other aspects of cross-platform solutions did not mention which design patterns they implemented. Thus, indicating the design pattern is insignificant when evaluating a cross-platform solution.

5.6 Sources of error

Before the thesis, the author of the thesis had no previous experience in developing Flutter or Android applications. This may result in unknown inconsistencies which could have affected both the application performance and application appearance. Thus, affecting the results of the thesis.

A more experienced developer might develop the sample applications to be more compact in terms of application size and more efficient in terms of application startup. However, it is doubtful it would affect the results in terms of performance.

Firstly, the significant difference in the application size is due to the overhead of the Flutter engine. No matter what experience a developer has developing Flutter applications, the engine is still embedded. Secondly, it is not likely a more experienced developer could affect the startup time significantly, as it also most likely due to Flutter internal workings.

5.7 Sustainability & Ethical Aspects

Investigation and evaluating application size is important from a socioeconomic point of view. It is known that application size is more important in emerging markets, where the users' devices might have limited persistent storage or unstable Internet connection. If applications grow too large, it might inhibit the usage of the application for the users in the emerging markets. Or even worse, completely deny the users the opportunity of access to certain applications.

Chapter 6

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.

For user perception, a study was conducted using the User Experience Questionnaire (UEQ). The aim of the study was was understand user perception and see if there were any significant differences between native and Flutter developed applications. The result seems to suggest there is no significant difference between a Flutter-developed application and a native application. There is a high uncertainty here, due to the few amount of participants in the study. More participants in the study would likely increase the confidence of the conclusion, Less likely, more participants might change the outcome, thus indicating a significant difference between the applications. However, the t-test performed seem to indicate that there wouldn't be any significant difference.

Chapter 7

Future work

From a research perspective, it would be beneficial to further evaluate Flutter with sample applications that include resource-intensive features. The outcome might be significantly different in terms of performance and user perception.

Furthermore, Flutter seems to have an unusual approach by bypassing the native libraries and not use an abstraction layer as other cross-platform solutions might. Focusing more on this aspect might give insights that could be beneficial to the next cross-platform solution. Is there any empirical data that supports this implementation?

Further investigation and evaluation of the Flutter engine might be relevant. It would most likely be valuable for the maintainers of Flutter. With more insight and deep knowledge about the engine, one might reduce the engine size or initialize it faster. Thus, reducing the performance gap between Flutter applications and native applications.

Bibliography

[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 Technologies*. 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 Science and Applications*. Springer, 2020, pp. 571–575.

[11] Young Hoon Kim, Dan J Kim, and Kathy Wachter. “A study of mobile user engagement (MoEN): Engagement motivations, perceived value, satisfaction, and continued engagement intention”. In: *Decision support systems* 56 (2013), pp. 361–370.

[12] Niclas Hansson and Tomas Vidhall. “Effects on performance and usability for cross-platform application development using react native”. MA thesis. Linköpings universitet, 2016.

[13] R. Nunkesser. “Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development”. In: *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. 2018, pp. 214–218.

[14] Andreas Biørn-Hansen, Tor-Morten Grønli, and Gheorghita Ghinea. “A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development”. In: *ACM Comput. Surv.* 51.5 (Nov. 2018). ISSN: 0360-0300.

[15] 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.

[16] Tim Majchrzak and Tor-Morten Grønli. “Comprehensive analysis of innovative cross-platform app development frameworks”. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017.

[17] 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.

[18] Rasmus Fredrikson. “Emulating a Native Mobile Experience with Cross-platform Applications”. MA thesis. KTH Royal Institute of Technology, 2018.

[19] Robin Abrahamsson and David Berntsen. “Comparing modifiability of React Native and two native codebases”. MA thesis. Linköpings universitet, 2017.

[20] Andreas Biørn-Hansen et al. “An empirical study of cross-platform mobile development in industry”. In: *Wireless Communications and Mobile Computing* 2019 (2019).

[21] M. Latif et al. “Cross platform approach for mobile application development: A survey”. In: *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. 2016, pp. 1–5.

[22] L. Delia et al. “Multi-platform mobile application development analysis”. In: *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*. 2015, pp. 181–186.

[23] Anthony I. Wasserman. “Software Engineering Issues for Mobile Application Development”. In: *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. FoSER ’10. Santa Fe, New Mexico, USA: Association for Computing Machinery, 2010, pp. 397–400. ISBN: 9781450304276.

[24] Esteban Angulo and Xavier Ferre. “A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX”. In: *Proceedings of the XV International Conference on Human Computer Interaction*. Interacción ’14. Puerto de la Cruz, Tenerife, Spain: Association for Computing Machinery, 2014. ISBN: 9781450328807.

[25] L. Delía et al. “Approaches to mobile application development: Comparative performance analysis”. In: *2017 Computing Conference*. 2017, pp. 652–659.

[26] Bonnie Eisenman. *Learning react native: Building native mobile apps with JavaScript*. " O’Reilly Media, Inc.", 2015.

[27] Peter Späth. “Your First Kotlin Application: Hello Kotlin”. In: *Learn Kotlin for Android Development: The Next Generation Language for Modern Android Apps Programming*. Berkeley, CA: Apress, 2019, pp. 1–14. ISBN: 978-1-4842-4467-8.

[28] Peter Späth and Jeff Friesen. “Getting Started with Java”. In: *Learn Java for Android Development: Migrating Java SE Programming Skills to Mobile Development*. Berkeley, CA: Apress, 2020, pp. 1–29. ISBN: 978-1-4842-5943-6.

[29] Wallace Wang. "Understanding iOS Programming". In: *Beginning iPhone Development with Swift 5: Exploring the iOS SDK*. Berkeley, CA: Apress, 2019. ISBN: 978-1-4842-4865-2.

[30] Erik Sorensen and M Mikailesc. "Model-view-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technology". In: *MegaByte Journal* 9.4 (2010), pp. 1–19.

[31] 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.

[32] Tian Lou. "A comparison of android native app architecture–mvc, mvp and mvvm". In: *Eindhoven University of Technology* (2016).

[33] 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.

[34] Thomas J. Quirk. "Two-Group t-Test of the Difference of the Means for Independent Groups". In: *Excel 2019 for Engineering Statistics: A Guide to Solving Practical Problems*. Cham: Springer International Publishing, 2020, pp. 85–113. ISBN: 978-3-030-39278-9.

[35] 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.

[36] Ton J Cleophas and Aeilko H Zwinderman. "Testing Parallel-Groups with Different Sample Sizes and Variances (5 Parallel-Group Studies)". In: *Machine Learning in Medicine -a Complete Overview*. Cham: Springer International Publishing, 2015, pp. 471–473. ISBN: 978-3-319-15195-3.

[37] Martin Schrepp. "User experience questionnaire handbook". In: *All you need to know to apply the UEQ successfully in your project* (Dec. 2019).

[38] Michiel Willocx, Jan Vossaert, and Vincent Naessens. "Comparing performance parameters of mobile app development strategies". In: *Proceedings of the International Conference on Mobile Software Engineering and Systems*. Austin, Texas: Association for Computing Machinery, 2016, pp. 38–47. ISBN: 9781450341783.

[39] Michiel Willocx, Jan Vossaert, and Vincent Naessens. "A quantitative assessment of performance in mobile app development tools". In: *2015 IEEE International Conference on Mobile Services*. IEEE. 2015, pp. 454–461.

[40] Atikant Singh, Aakriti V Agrawal, and Anuradha Kanukotla. “A method to improve application launch performance in Android devices”. In: *2016 International Conference on Internet of Things and Applications (IOTA)*. IEEE. 2016, pp. 112–115.

Appendix A

User Experience Questionnaire

Vänligen bedöma produkten nu genom att kryssa en cirkel per rad .

	1	2	3	4	5	6	7		
Irriterande	○	○	○	○	○	○	○	Njutbar	1
Obegriplig	○	○	○	○	○	○	○	Begriplig	2
Kreativ	○	○	○	○	○	○	○	Tråkig	3
Lätt att lära sig	○	○	○	○	○	○	○	Svår att lära sig	4
Värdefull	○	○	○	○	○	○	○	Värdelös	5
Tråkig	○	○	○	○	○	○	○	Spännande	6
Ointressant	○	○	○	○	○	○	○	Intressant	7
Oförutsägbar	○	○	○	○	○	○	○	Förutsägbar	8
Snabb	○	○	○	○	○	○	○	Långsam	9
Uppfinningsrik	○	○	○	○	○	○	○	Fantasilös	10
Hindrande	○	○	○	○	○	○	○	Stödjande	11
Bra	○	○	○	○	○	○	○	Dålig	12
Komplicerad	○	○	○	○	○	○	○	Enkel	13
Möter inte behov	○	○	○	○	○	○	○	Möter behov	14
Bakåtsträvande	○	○	○	○	○	○	○	I framkant	15
Inte tilltalande	○	○	○	○	○	○	○	Tilltalande	16
Säker	○	○	○	○	○	○	○	Inte säker	17
Motiverande	○	○	○	○	○	○	○	Omotiverande	18
Möter förväntningar	○	○	○	○	○	○	○	Möter inte förväntningar	19
Ineffektiv	○	○	○	○	○	○	○	Effektiv	20
Tydlig	○	○	○	○	○	○	○	Förvirrande	21
Opraktisk	○	○	○	○	○	○	○	Praktisk	22
Strukturerad	○	○	○	○	○	○	○	Rörig	23
Estetisk	○	○	○	○	○	○	○	Oestetisk	24
Användbar	○	○	○	○	○	○	○	Inte användbar	25
Konservativ	○	○	○	○	○	○	○	Innovativ	26

TRITA -EECS-EX-2021:756