A Comparison of Performance and Looks Between Flutter and Native Applications

When to prefer Flutter over native in mobile application development

Matilda Olsson

June 13th, 2020

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Department of Software Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Software Engineering. The thesis is equivalent to 10 weeks of full-time studies.

Swedish Title: En jämförelse mellan Flutter och native applikationer - När ska man välja Flutter över native för utveckling av mobila applikationer

Contact Information:
Author:
Matilda Olsson
E-mail: maoo17@student.bth.se

University advisor:
Andreas Arnesson
E-mail: andreas.arnesson@bth.se
Department of Computer Science (DIDA)

External advisor:
John Söderqvist
E-mail: john.soderqvist@consid.se
Consid Karlskrona

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se/dipt
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

A mobile application has to be able to keep up with heavy demands to compete with all the new applications that are developed each day. Good performance and nice visuals are base requirements for the development of mobile applications. There are many options for tools when developing and one of these choices is a native application, which is said to have better performance and suitability to the mobile environment. Another choice is a tool which requires only one code base for multiple platforms and is therefore easier to maintain. Flutter is an open-source User Interface (UI) toolkit created by Google that can create cross-platform applications with one code base while said to maintain the aspects of looking native.

This paper explores how Flutter compares to native applications, which are currently seen as superior in mobile behaviour and performance. An experiment was conducted to test how Flutter as a cross-compiler compared to two native applications made of kotlin and Android studio and swift and XCode, in terms of CPU performance. A survey was created to see if there was a difference in the perception of users with regards to appearance and animations. A literature study was conducted to strengthen the results from the experiment and survey and to give a background to the subject.

Flutter is a new tool and it continues to grow incredibly fast. Conclusions are drawn that a Flutter application can compete with a native application when it comes to CPU performance, but is not as developed in the animation area. Flutter does not require complex code for creating a simple application and uses significantly less lines of code in development compared to native. The final conclusion is that Flutter is best to use when building smaller to medium-sized applications, but has a potential to grow to overcome its current drawbacks in the animation department. Further examination of the areas examined in this paper is needed in order to ensure and strengthen the results.

Keywords: Flutter, Cross-platform, Native, Performance, Mobile Applications.

Contents

1 Glossary

SDK - Stands for Software Development kit. A toolkit that helps making the development of software applications easier. They often include debugger, compiler and software framework.

Cross-platform - Often refer to a product(in this case application) that can run on different platforms.

Native application - Native in the application world refers to the development of an application that is specifically developed for an operating system (OS).

Swift - A Programming language developed by Apple Inc which is used to create iOS native applications in this project

Kotlin - A Programming language developed by JetBrains which is used to create Android native applications in this project.

Dart - A Programming language developed by Google which is used to create Flutter applications for iOS and Android in this project.

AOT - Ahead of Time Compilation

JVM - Java Virtual Machine

UI - User Interface

SIL - The Swift Intermediate Language

LLVM - Low level virtual machine. A Collection of compiler and tool chain technologies which are reusable and modular.

LLDB - Low level debugger. A debugger that is the default debugger in XCode and part of the LLVM collection.

2 Introduction

The purpose of this paper is to evaluate Flutter as a UI tool for creating mobile applications. A method that is used for creating mobile applications is by creating them for a specific platform from the beginning [1]. Another method is to write a code base that can be compiled to several types of platforms. It is a method called cross-platform and is a popular method because of how flexible and fast a mobile application can be created [2].

Choosing between native and cross-platforms is often a question about money and the proper way of developing. Flutter is an open-source UI toolkit that uses the language Dart that can create mobile applications with a single code base and compile the code into both Android and iOS [3][4]. It was created by Google year 2018 and can according to themselves, create applications that inherits the same type of look, feel and performance as if they would have been developed as a native mobile application [5]. This paper contains a study where Flutter is compared to native applications in different aspects such as performance of the CPU, visuals, code complexity and code needed to perform its designated task. Many of the terms that are mentioned in this paper are explained in section 6.

2.1 Background

The study in this paper was done in collaboration with the Company Consid. They are a company with multiple offices throughout Sweden and work with consulting in the software engineering field, developing products such as web applications, mobile applications and other software. Their customers normally chooses native applications and often have strong opinions of which technique and solution they want from the beginning. By researching Flutter, Consid will gain more knowledge of Flutter to be able to give a better motivation to if it could be a good candidate for their customers or not.

To be confident in their recommendations, Consid requested to see the performance results from the comparison between native and Flutter. They also requested the look and feel to be compared together with the code structure and amount of code that it takes to create a Flutter application.

There will be a creation of 4 applications with the help of 3 programming languages(Kotlin, Dart and Swift). Flutter with Dart as the programming language, will produce two of these applications. The other two applications will be made with Kotlin and Android Studio as well as Swift and XCode.

2.2 Purpose

Despite its recent release, Flutter has a lot of talk around it and many praises the tool for how good it is even though no one seems to actually use it in action [6][7]. Even though many praises the tools qualities, there are still little

knowledge on where it fits in between the native development and the cross-platform tools.

A motive for this paper is to understand if Flutter is a good candidate as a cross-compiler, how close it comes to competing with native in CPU performance and if it performs better, worse or the same as a native application. Where can a developer set the preference whether to use Flutter over native? This is going to be measured by a survey, literature study and an experiment where the results will be weighted together when creating an opinion about how close Flutter comes to competing with native.

A Problem that may have occurred, is that there might have been too few academical papers on Flutter that can strengthen the underlying support of this paper. Another challenge is that the survey is not sufficient because of the limitations in doing a live survey or because of misunderstood questions.

Development of all the applications follows the guidelines of the respective documentations for mitigating the risks of the compared applications being too different in development approaches.

2.3 Contribution

This study will delve deeper into how Flutter works as a tool for creating mobile applications. Furthermore, the study will answer the research questions to bring value for Consid and for their customers. This is for them to be able to choose new alternatives to native applications.

By researching the performance, it gives Consid an understanding of how or if Flutter can be useful to their customers. Studying the look and feel will in addition to performance metrics, give an insight of how well Flutter can perform and be presented when discussing alternatives to native applications. The structure and amount of code together with the look and feel result, will reveal if Flutter can keep up its small code base while maintaining a closeness to the same visuals as a native application as well as perform equally or better to a native application.

3 Related Work

Related work that has been found for the subject revolving Flutter and comparisons with Native are close to none except for an article by Coninck that was based on a published paper which could not be found. There are however, many academical papers on performance differences between cross-compilers. One of these papers (Heitkötter et al., 2012) discusses the differences between cross-platforms. The papers results showed that cross-platforms were better for shorter time and budget. Native were preferred when interacting with the phone's integrated system [8][6].

Corral et al., writes about a study made on performance for PhoneGap versus native. The result showed that the cross-platform applications were slower than native in 7/8 performance tests. However, this performance study was specifically targeted on android devices [9].

A paper written by Amatya 2013, brings up the differences in cross-platforms and native by researching the platform, development environment and code base. Amatyas conclusion was that native is a good fit for heavier applications but is not always the most suitable choice for all applications because of it being cost heavy and more time consuming than cross-compilers [10].

Another paper that contains a performance study on a cross-platform versus native is a study written by Axelsson and Carlström. It studies React native in comparison to a Swift and a Java application. The results that was presented in the study were that the performances between the application builds, were the same with minor differences. The area were the differences showed the most, was the animations [11].

As seen in earlier mentioned related work papers, many uses React Native or Phone Gap for comparing with native. Guerra carried out a comparative study on cross-platform frameworks and native. In the study, he measured the execution time amongst other metrics and the results showed that Flutter had a significantly faster execution time than the other cross-platforms which were React Native and Ionic. This can show to be useful for this paper's study since the earlier performance results are based upon React Native, but is not an official publication and the original paper could not be found [12].

The conclusion of reading related work was that there are many papers researching cross-platform and native but there is a lack of performance testing between native and cross-platform approaches since most only compare code and flexibility. There was however one paper written by Dalmasso et. al 2013 that studied how cross-platforms performed against each other. The authors looked at CPU- and memory usage and of different cross-platforms. Focus were laid on the differences in these metrics depending on which dependencies the application were using [13].

4 Goal

There is a good amount of academical papers that researches the differences in aspects of native and cross-compilers such as react native, but very few that mentions Flutter. This could be because it is a rather new tool with its first stable release in 2018 [14] [1][3].

The goal of this report is to see how closely Flutter behaves to native without outweighing the beneficial sides that comes with cross-platforms such as developing speed and easier maintainable code base.

4.1 Focus

The study will look at the aspects of Flutter and how it performs, structures code and manages looks of the application. This involves the CPU usage and the grade of confirmability to native behaviour.

This report will not look at other metrics regarding performance. Run time CPU usage were the metric that Consid specified for the study. They regarded the metric as the one that would be the most interesting when using and developing a mobile application. Build Process CPU usage was not researched because Consid did not mention this as an area that they wanted to explore further.

4.2 Research Questions

- RQ1: How does Flutter differentiate in code size and how complex code is needed in comparison to native built applications?
- RQ2: How well does Flutter perform in run time CPU usage compared to native applications on Android and iOS?
- RQ3: How much does Flutter and android native differentiate in terms of look and feel for the application users?

4.3 Research questions explained

RQ1: How does Flutter differentiate in code size and how complex code is needed in comparison to native built applications? Code size and code complexity is two things that is important for the development of a mobile application. This question will help to understand the development differences that there are in code size and complexity. This is done to create an understanding of how easy the code base languages are to learn and how much code is needed to obtain the wanted results.

RQ2: How well does Flutter perform in run time CPU usage compared to native applications on Android and iOS? This question will investigate if Flutter can perform at the same level of CPU performance as native applications. CPU performance is currently one of the reasons why native is considered a better option to other solutions.

RQ3: How much does Flutter and android native differentiate in terms of look and feel for the application users? The look and feel of an application is the first thing a user experiences when using an application. This

question will delve deeper into how much difference there is between a Flutter and a native application in terms of looks and feel.

Since both Flutter and Android is developed by Google, it would be have been a better option to compare iOS to Flutter. Instead, Android was chosen because of unpredictable events regarding the iOS technology that was to be used.

5 Method

To find the results for the three research questions and the under laying support for this paper, three different methods were carried out.

A literature study was conducted to give an understanding to the author and the reader how Flutter works. It contains information on how the three tools and languages that was used in the experiment, works in development. The research that was found in the literature study acted as a support for methods used in the experiment.

To answer RQ2, an experiment was used. The experiment included four applications that were made in Dart with Flutter, Kotlin with Android Studio and Swift with XCode. Two of them were made of the single Flutter and Dart code base. It was carried out to find out how Flutter compares in the ares of CPU usage compared to native applications.

RQ1 was answered by studying the authors code review for the development of the experiment applications. It was done after the development to ensure that the code bases were complete before examining. Development time was measured by taking the time that each of the applications took from that of generating the starting layout to compiling and running the final version. Measurements of lines, files, size and application size were taken from the finished projects and applications.

A survey was created and sent out to answer RQ3. It had questions revolving look and feel between the two earlier created android applications. This was done to see if the survey takers could notice any difference between them and to see how much difference it would make for them as users.

5.1 Literature Study

The literature study for this paper was executed to give the reader a better understanding of the cross-platform and native applications. Studying how Swift/iOS, Kotlin/Android and Dart/Flutter handles development and compiling. It contains earlier studies on cross-platform and native comparisons.

The peer reviewed publications that were used for this paper comes from the

databases: Google Scholar, Diva, IEEE and BTH Summon. The following keywords and sentences were used:

- Cross-platform vs native
- Google Flutter
- Cross-platform performance
- Native
- Cross-platform tools
- Mobile application performance

Another search word that was used in the search for material on performance was "PhoneGap performance". By looking at the earlier related work, the majority of studies found on cross-platform comparisons uses PhoneGap which makes it a relevant search word for finding comparative studies.

The tools official pages were used for primary information about documentation and code standards. Online technology articles were used to find general information on differences between the ways of creating applications. Multiple articles was used to ensure a strengthening of the information that came from online articles.

With Flutter being fairly new, it was hard to find a good amount of publications. This is why most of the studies and articles specifically about Flutter are taken from non peer reviewed publications. However, the priority was still put on finding peer reviewed publications in the first place. Information on cross-platforms versus native and books about Flutter was taken from the earlier mentioned databases.

Snowball sampling was used on the first scholary database results that showed up from searching with the search words. All of the paper types that was found in the scholarly databases were used to conduct the literature study.

5.2 Experiment

In this report an experiment was carried out to answer RQ1 and RQ2. This section presents the experiment and how it was prepared, its goals and the process itself. It is done to give the reader a better understanding of how the experiment was executed and planned.

Due to outside influence, the method had to be changed significantly from the prior method of this thesis work. In summary, promised software that were to be used in the project was not delivered. Therefore adjustments had to be made that changed the method and execution significantly. This will be further analysed in the analyse section of the report.

Battery consumption is an important metric for mobile applications CPU usage. The measurement of battery consumption was not possible due to the measuring tools needing a connection through USB cord which made the mobile phones charge automatically while performing the tests.

Four applications were made with Flutter(which creates two applications), Kotlin and Swift. The applications were created to look like each other and with code that followed according to each documentation.

A relatively simple layout was chosen because of the time constraint of the project and to be able to handle complex code. The application layout consisted of a navigation bar and two page views "Dashboard" and "Notifications". Dashboard were filled with a picture of a salad and had the headline "Spring Salad Recipe" while notifications had a simple list of labels with the word "Notification". Each view had an app bar at the top with the view name.

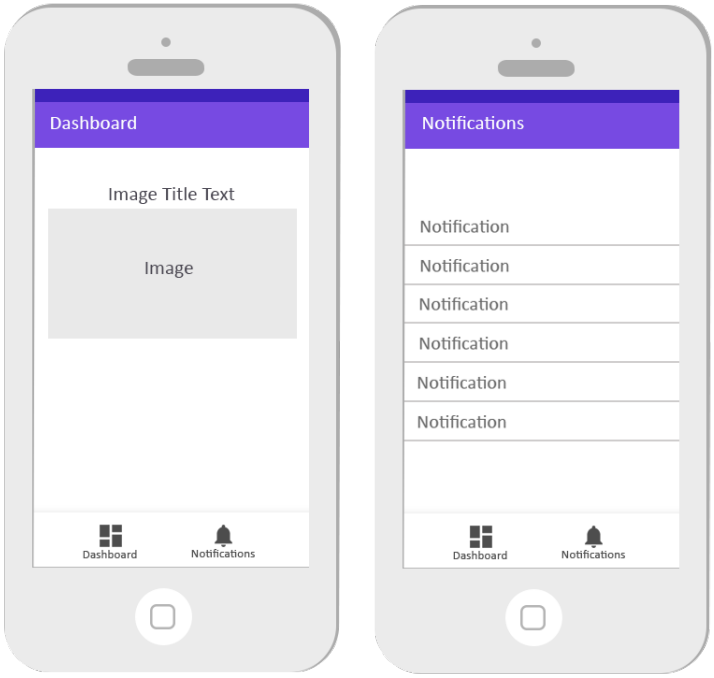


Figure 1: Example application layout appearance

For this system to be able to represent a real life system, the UI and flow of the application were inspired by applications such as: Tasty application and the

Notification history application. Navigation bars exist in most of the applications today that has content and is not dependant on the amount of items that exist in the bar itself, but the actual function and look of the navigation bar. The example application was inspired from the "Tasty" application where the same look of the navigation bar exists as well as the recipe part of the example application. A similar listview to the one that is used in the example application design, can be found in the "Notification history" application.

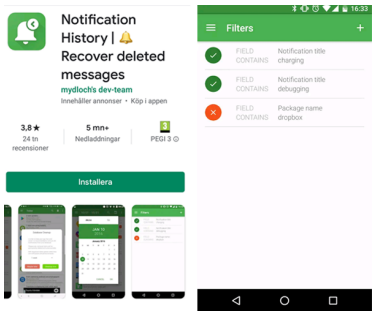


Figure 2: Notification History application

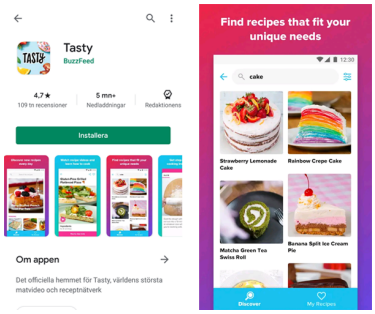


Figure 3: Tasty application

5.2.1 Experiment Goal

The goal with this experiment was to find out the difference between the Flutter and the native applications in terms of run time CPU usage.

Another area that was studied, was the amount of code that each development environment required to create the desired looks and functionality of the applications. This was a side to side measurement and gave an insight of how the native code bases compared to the Flutter code base.

5.2.2 Execution

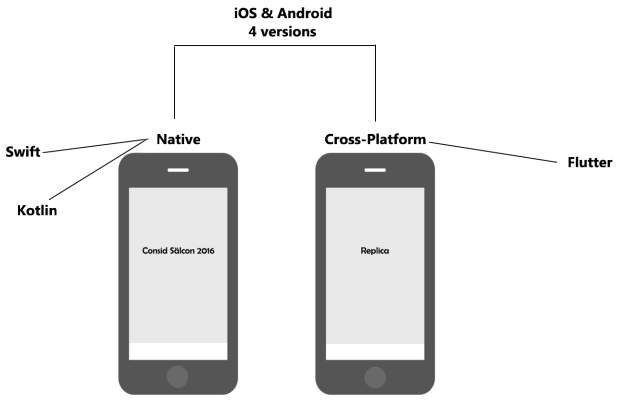


Figure 4: Graphic representation of the applications and their technology

Four applications in Flutter, kotlin and swift were created and built. Each development environment was setup accordingly to their documentation:

- Flutter [15]
- Android - Android Studio [16]
- iOS - Xcode [17]

The CPU usage was measured three times manually per application build and the highest, lowest and mean values were written down. Values that were higher than 0 was chosen for the lowest, as the CPU showed 0 when the applications were passive. The tools that were used to measure were: Android studio profiler (Android) and XCode instruments (iOS). Because the output from the measurements instruments only showed as graphs, sampling was used to determine mean value and ease measurement. Standard deviation was then calculated for each set of values. The collected values and mean of each test case were gathered in a summarised table.

The author created the applications and measured development time without prior experience with application development. The first prototypes were used for the experiment.Code complexity was measured by the author through the code review.

Devices that were used for the experiment were one iPhone 7 and one Android Samsung S7. These mobiles were chosen to match each others performance for more precise results and because of easy accessibility. Exact device specifications that were used for this experiment can be found in Appendix A under "Device specifications".

To create a base that would align to each language's standards, terminal commands and templates were used to generate a code base that contained a navigation and two connected views. This was done to follow the documentation as much as possible. Package and tools can be found in Appendix A under "Tools and packages".

Default margin was used when positioning parts such as images and lists. This means that the recommended margin was used for each development environment.

The navigation route below was followed manually when measuring the CPU usage. It was done to generate a good amount of data of the applications functionality and to ensure all the test measurements to follow the same guidelines.

1. Start the application
2. Wait for Dashboard view to load
3. Navigate to Notifications view
4. Navigate back to Dashboard view
5. Navigate to Notifications view
6. Scroll up and down in list 4 times
7. Return to the Dashboard view

5.2.3 Preparation of the Devices for testing

The devices were prepared by doing the following before the measures:

- Fully charging the battery
- Activating Flight mode
- Clearing the processes by restarting the phone
- Restarting the application before each measurement

5.2.4 Hypothesis

The hypothesis is that Flutter performs equal to or better than a native based application in this particular study. This theory is based upon the earlier study

made by Coninck where he performed an experiment where Flutter was compared to native android, xamarin forms, native iOS and react native which showed that Flutter had a lower CPU usage [6].

Taking a look at studies made between native and cross-platforming without specifying Flutter, the result shows that the cross-platforms are to be preferred because of their many advantages in development speed and that the difference in performance are small [11][18].

5.3 Survey

To answer RQ3, a survey was created and sent out to people who worked or had been educated in the software industry. Two of the applications android native and android flutter were compared without the participants knowing which one was android native or android Flutter.

The look and feel of the application was measured by the survey takers answering questions with a 1 to 5 scale where 1 could be "I don't use a phone" and 5 would then be "I use a phone everyday" together with yes/no questions as well as input fields for explanations in which the user had to explain the choices based on the looks, behaviour and animations of the applications. This type of answering was chosen because it produces easier measured answers than if the answers would have been in free form.

5.3.1 Preparation

Preparing for the survey, the earlier created android applications in the experiment was used for each participant to analyze and compare. The following setup was done to identify each application for later use in the results:

App A - Android native
App B - Flutter Application

5.3.2 Survey Goal

The survey goal was to find out how or if Flutter feels and looks different from a native application. This was done to see how much a user notices the differences and what impact it has for them. Results from this survey were then used together with the results of the experiment study on performance to conclude on the differences between native and Flutter.

5.3.3 Target group

The target group for this survey will mainly consist of employees at Consid, students and teachers at Blekinge Institute of Technology. These people have a high probability to have knowledge about web techniques and a basic design knowledge as well as using many different applications daily.

6 Literature Review

This section will give an insight in the backgrounds of Flutter, Swift and Kotlin, as well as the development and building environments that were used in the experiment. It will provide support for RQ1 and RQ3.

The goal of this literature review, is to help the reader and author understand how each language and environment works and what differentiate them in terms of functioning, management of looks and development.

6.1 Native mobile applications

The meaning of native in the field of mobile applications refers to applications that are built to run on a specific platform or OS. There are numerous languages that can be used for building native mobile applications and a few of examples of these are: Kotlin, Java and Swift. Mainly, developers has been focusing on the native building of applications because of the customization's that is enabled for the native devices such as camera access [19].

6.1.1 Native implementation of UI

One thing that is associated with native mobile applications, is that there are more fluid looking animations and easier integration with the mobile technology. Native applications inherit the targeted platforms looks and apply it to their appearance, this is called Native UI. This allows the native applications to behave and look more accordingly to the "native" system to give the user a more relatable experience to the mobile platform OS itself [20].

6.2 Cross-platform mobile application development

Cross-platforming refers to a product or software that can be used on another platform than what it was developed for. In app development the principle of cross-platforming is to build and maintain only one code base, which is the alluring part of using it since it saves time in development compared to native that is limited to only one platform per code base. Examples of cross-platform frameworks/tools are Flutter, React Native and Ionic [21][22].

6.3 Flutter

Flutter is a mobile SDK and UI tool that is developed by Google, based on the programming language Dart and officially released in 2018. The developers' goal is to deliver applications that comes as close to the performance and look and feel of native applications [23][24].

6.4 Dart

Dart is a programming language created by Google. It is an client-optimised language that can be used on multiple platforms. It has C-style syntax and is object oriented using classes. Dart can be compiled into both JavaScript and native based code [25].

6.4.1 Flutter/Dart UI management

Flutter uses **widgets** as the main concept within in the code. Widget is the nickname for every component part that is built in Flutter. This could mean a box or a text that is referred to as a widget. A noticeable part of the widgets is that they are created by the Flutter developers to look native and developers are able to fully customize these to their liking. Flutter uses a material components library by default which lets the developer use components that are ready from the beginning, which is a concept web technologies often work with [26][27][28].

Flutter has a development functionality called hot reloading which means that when changes are made, they are injected into the dart Virtual Machine and Flutter rebuilds the structure and let you view the changes that was made faster [5][27].

6.4.2 Flutter compiling

The way Flutter works when running on Android is by compiling the developers written Dart code to native with the help of AOT compiling. These together with x86 libraries that were created when compiling the dart code, are put into a runner and built as an APK. Flutter runs similar on iOS system, but instead the Dart code compiles into ARM library and is later put as a runner and built as an .ipa. Flutter themselves suggest the whole process is similar to how game engines works and that is how the developers explain it on their website [24][27].

6.4.3 Use and popularity

In a paper written by Mehdi Satei 2019, there is a section where the author discusses which programming languages and frameworks that are popular in the industry. Satei presents statistics from 2019 where Dart is on the rise of most wanted languages while Flutter is top 3 on most wanted frameworks [26].

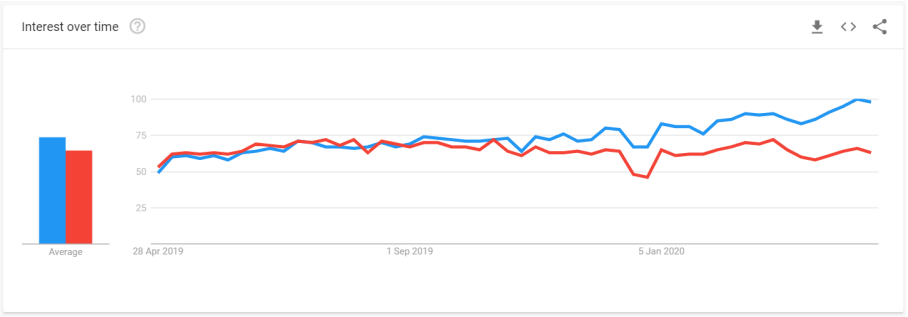


Figure 5: Graph of React Native and Flutter interest of search from Google trends https://trends.google.com/trends/explore?q=Flutter,React%20Native (React Native is red and Flutter is blue

Looking at the Google trends chart for Flutter in comparison to React Native, there is a difference starting in trends for searching which means that more people are taking interest in Flutter development.

Flutter is however, not used much when looking at the Stackoverflow survey of 2019 where Flutter is only used by 3.4% of all users and not even on the list for the professional developers use. 1.9% of the users liked Dart while for the professional developers use, Dart was not on the list. Both Flutter and Dart is however very highly ranked in the "Most loved" categories as 3rd and 12th [29].

In the Stackoverflow survey of 2020, Flutters use statistics has increased from 3.4% to 7.2% in the overall statistics, but stays off the list as before in the professional only statistics. Flutter stays in the 3rd position as the most loved framework/library/tool for 2020 [30].

Dart is still off the list of the professional use of Dart, but has increased from 1.9% to 4.0% since 2019. For the 2020 Stackoverflow survey, it has become the 7th most loved programming language [30].

6.5 Kotlin

Kotlin is a programming language that is made by Jetbrains. It is open source software statically typed language. The result when doing so would be a compiling error. Kotlin is fully compatible with Java and supported on the Android platform for creating Android applications [31][32].

6.5.1 Kotlin compiling

According to the documentation, Kotlin compiles into compatible bytecode for Java when targeted on the JVM. Kotlin is a versatile programming language that aims to be able to function on multiple platforms. If targeted to native,

Kotlin will go through the LLVM to produce specific code for the platform in question [33].

6.5.2 Kotlin/Android UI management

When designing an UI in Android together with Kotlin, the developer is inclined to use the Android view group system. The layout and UI elements can either be declared in XML or in code at run time. Drag and drop is available for the non-programmable layout creation. Both of these options needs to have the UI parts connect to the code and will need to be initiated upon creation in the onCreate method for the class layout. The objects in the layout is referred to as viewGroups and views. The views are called widgets which are can be a text object or an image. These are encapsulated by the viewGroups that are the layouts [34].

6.6 Swift

Swift is a programming language created by Apple Inc 2014. It was made to be able to work on multiple of platforms and aims to be a replacement to the C-languages. Swift is managed as a group of projects where it is split into: swift compiler, standard library, core libraries, LLDB library, swift package manager and Xcode playground support [35].

6.6.1 Swift compiling

Swift code is compiled down to machine code with the help of seven level steps in the compiler. The parser is ran at the beginning to check if there is any grammatical errors and show warnings. After the parser, a semantic analysis is carried out to see if its safe to compile the code without errors. Clang importer then imports clang modules that can be referred from the analyser. This is followed by something called a SIL generation and SIL guaranteed transformations. The first SIL runs another analysis on the code to improve optimization. The second SIL do data flow diagnostics to check so there are no uninitialized variables and results in a canonical SIL, meaning that it is a SIL that exists after the optimization and analyses has been done. There is an additional SIL step that is called the SIL Optimizations where extra high-level swift optimizations in additions to the ones before, are carried out. At the end there is an LLVM IR Generation which means that SIL is transformed into machine level code with the help of LLVM, which are compiler tools [36][37].

6.6.2 Xcode UI management

When developing a UI in Xcode, the Interface Builder is used to create storyboards. Scenes are used in storyboard to represent what is seen and what is happening on the device screen. Segues connects the scenes and holds upholds their relation. Scene objects can be dragged and dropped to create a new item

to view. These items and controllers can be connected to code manually or with the help of Xcode assistant that can create or generate code automatically [38].

7 Results

This section displays the results that were discovered through the experiment and literature review. Pictures of the final applications are found below.

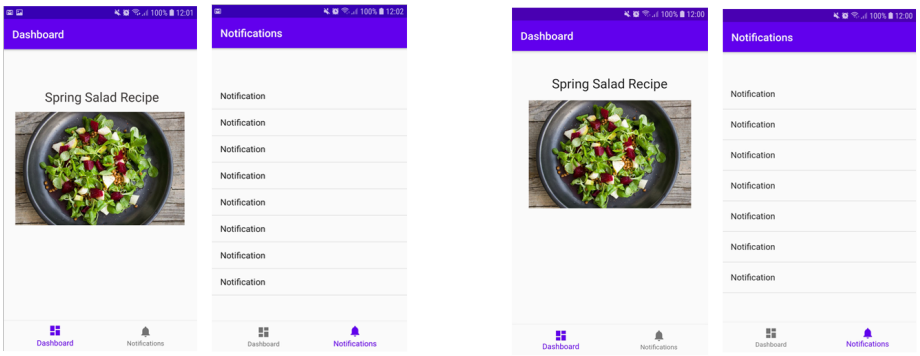


Figure 6: Images of final android native and android Flutter applications

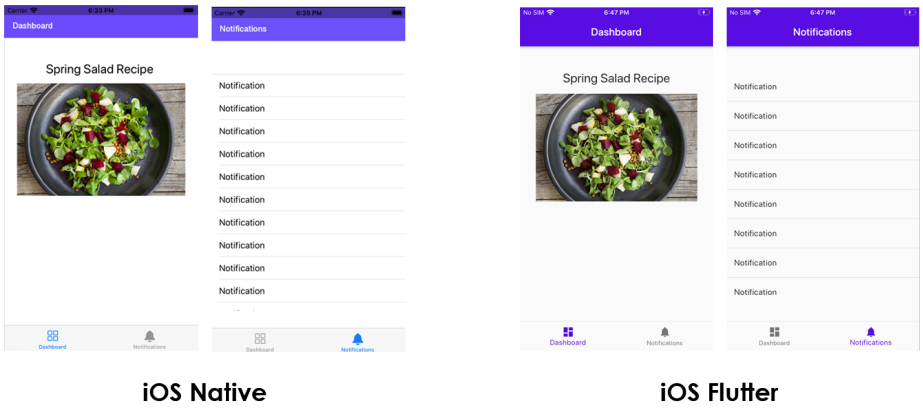


Figure 7: Images of final iOS native and iOS Flutter applications

7.1 RQ1: How does Flutter differentiate in code size and how complex code is needed in comparison to native built applications?

Looking at the results as a whole, Flutter wins the majority of most categories in the development area. There are however some differences that are interesting to take note of when comparing Flutter to native builds.

Type	lines of code	Code files that were needed for the application
Android native	217	9
Android Flutter	125	3
iOS native	363	6
iOS Flutter	125	3

Figure 8: Application lines of code and file count

Type	Total	Navigation Base	First View	Second View
Android native	12h	3h	2h	7h
iOS native	8h	2h	0.5h	5.5h
Flutter	6h	4h	1h	2h

Figure 9: Development time of each code base

7.1.1 Code size

As seen in figure 8, Flutter had the lowest amount of code lines and files that were needed in order to create the application. Native iOS had the lowest size of project files and app size but a significantly larger amount of code lines than the other builds. The native android had the most amount of files created and required lower amount of code lines than the iOS native.

7.1.2 Code Complexity

A part of answering Q3 is comparing the code complexity of the development code of each of the applications. The part of the application that was chosen for this was the creating of the notification list. It was chosen in particular because it contained the most code that was written and because the other view only featured an image and a title. The code that is shown in this section is only a part of the application code bases.

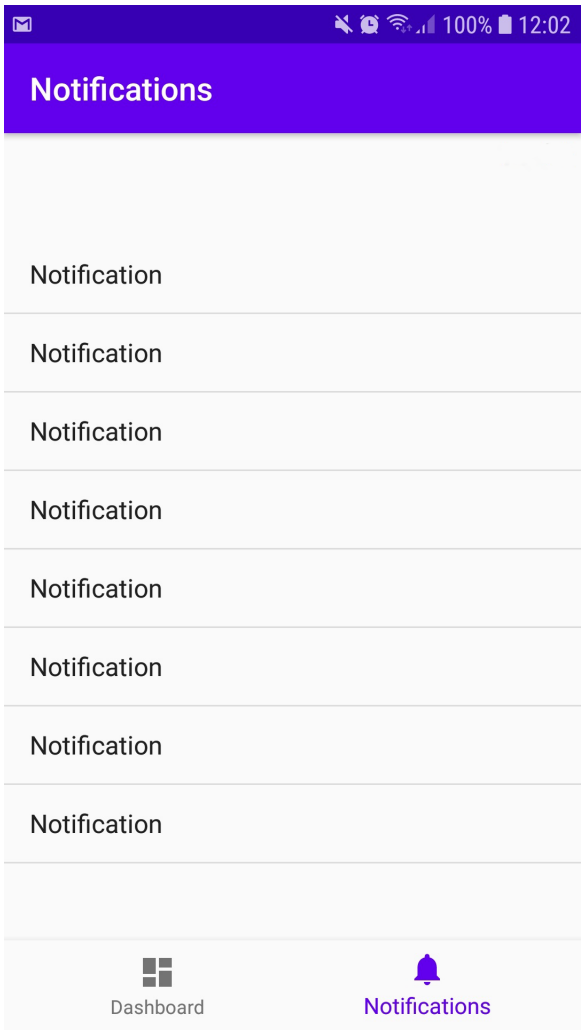


Figure 10: Picture showing notification list that was chosen for the code complexity part

```dart
import 'package:flutter/material.dart';

class NotificationPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
        child: Padding(
            padding: EdgeInsets.only(top: 50.0, bottom: 40),
            child: ListView.builder(
              itemCount: 20,
              itemBuilder: (context, index) {
                return Container(
                  decoration: BoxDecoration(
                    border: Border(
                      bottom: BorderSide(
                          style: BorderStyle.solid, color: Colors.black12), // BorderSide
                    ), // Border
                  ), // BoxDecoration
                  child: ListTile(
                    contentPadding: EdgeInsets.only(left: 15),
                    title: Text("Notification"),
                  ), // ListTile
                ); // Container
              },
            ))); // ListView.builder // Padding // Container
  }
}
```

Figure 11: Flutter code snippet showing the creation of notification list

Figure 11 shows how the Flutter code creates the visual layout and functional code in the same code. In the code image, there is a child parent relationship for the widget elements. The code shows that there is a widget that is built to return nested widgets. The child to the main containers Padding, shows the creation of a ListView Flutter widget. This view returns a ListView with 20 items and 20 container items.

```kotlin
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.LinearLayout
import android.widget.ListView
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProviders
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.navigation_bachelor_thesis.CardAdapter
import com.example.navigation_bachelor_thesis.CardItem
import com.example.navigation_bachelor_thesis.R
import kotlinx.android.synthetic.*
import kotlinx.android.synthetic.main.fragment_notifications.*


class NotificationsFragment : Fragment() {

    private lateinit var notificationsViewModel: NotificationsViewModel

    override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
        savedInstanceState: Bundle?
  ): View? {
        notificationsViewModel =
            ViewModelProviders.of(this).get(NotificationsViewModel::class.java)
        val notificationList = Array<String>(20, { i -> "Notification" })
        val root = inflater.inflate(R.layout.fragment_notifications, container, false)

        val listView: ListView = root.findViewById(R.id.products)
        listView.setAdapter(
            ArrayAdapter<String>(
                activity!!.applicationContext,
                android.R.layout.simple_list_item_1, notificationList
            )
        )


        return root
    }
}
```

Figure 12: Android native code snippet showing the creation of notification list

In figure 12, the android Kotlin application code declares the view model and import necessary items for the code. Everything happens in the onCreateView function, which inflates the XML layout for the notification list and creates an array with the same strings "Notification" that it injects as data to the already existing ListView XML target with the id "products".

The android native code has a lot of environment specific variables that a developer have to take in for consideration.

```swift
import UIKit

class SecondViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {

    @IBOutlet weak var table: UITableView!

    let items = Array(repeating: "Notification", count: 20)


    override func viewDidLoad() {
        super.viewDidLoad()

        self.table.register(UITableViewCell.self, forCellReuseIdentifier: "PlainCell")
    }


    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return items.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = self.table.dequeueReusableCell(withIdentifier: "PlainCell")! as UITableViewCell
        cell.textLabel!.text = self.items[indexPath.row]
        return cell
      }

}
```

Figure 13: Image of iOS code

As shown in figure 13, the iOS swift fetches the table from the corresponding storyfile and creates an array within, which it registers a tablecell(item) with the name "PlainCell". The TableView function checks the number of items in the created array "items" and returns a cell to the TableView with the text from the array. All this happens in the class for the specific controller in which the notification view exists.

This code is relatively short for its purpose and it is easily structured for a beginner in mobile development. There are language specific parts of the code but otherwise it could probably be understood by someone who have knowledge in other languages.

7.1.3 Development time

In figure 9 Android native had the longest developing time in total which was 12 hours, followed by the iOS native with 8 hours and lastly the Flutter which had the shortest of 6 hours.

In the development of both native applications, the use of drag and drop can be utilized for faster development. This is useful until a certain point when

the dropped elements need to be connected with code which was harder to do than generate the layout through direct code. This makes it easier to develop because the fact that the layout is always visible without the need of building it. Corresponding development function in Flutter is the hot reload function which lets you build the application and be able to reload it based on new features added.

7.2 RQ2: How well does Flutter perform in run time CPU usage compared to native applications on Android and iOS?

Type	Language	Highest	Lowest	Mean	Standard Deviation
Native iOS	Swift	92.7%	14.3%	32.9%	13.75360872
Flutter iOS	Dart	101.7%	18.8%	35.3%	18.00680891
Native Android	Kotlin	34.6%	1.0%	11.7%	6.88638675
Flutter Android	Dart	32.3%	1.0%	13.2%	9.29106696

Figure 14: Table showing summarized results of CPU measurement. Individual results for each development platform can be found in Appendix D

Using both the native and flutter builds for running CPU usage tests, they showed different results in terms of platform. Both the iOS applications showed higher CPU usage in the beginning as seen in figure 14.

Figure 14 shows the Android versions differentiated little. The Android Flutter application had lower max CPU usage but the native android had a lower mean value and both had the same lowest value. The android applications overall, had a high CPU performance in the beginning but not as high as the iOS applications.

7.3 RQ3: How much does Flutter and android native differentiate in terms of look and feel for the application users?

The survey was active during the time period 04/03-2020 to 17/04-2020 and was sent out to people that mainly works or are educated in the IT industry. There were a total of 39 people that answered the survey during the time period it was available.

In the survey, the application A was the native and application B was the Flutter application. A full view of the questions and a spreadsheet of all the answers can be found in appendix C.

7.3.1 Experience with the operating systems

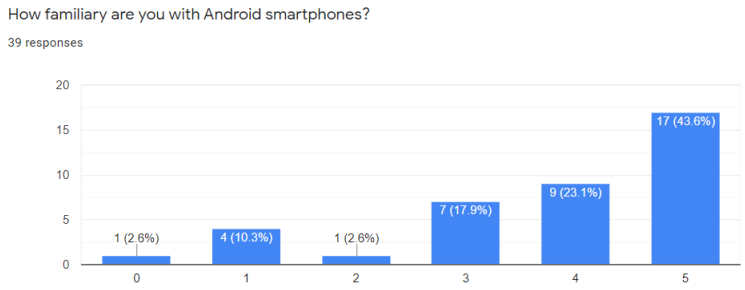


Figure 15: Diagram showing survey takers knowledge of android

In figure 15, only one of the survey takers had never used an android nor owned one. Most of the takers had 3-5 in experience with android applications beforehand. This meant most of the users were familiar with the android which was an advantage since the pictures and animations were android.

7.3.2 Application looks, animation and behaviour

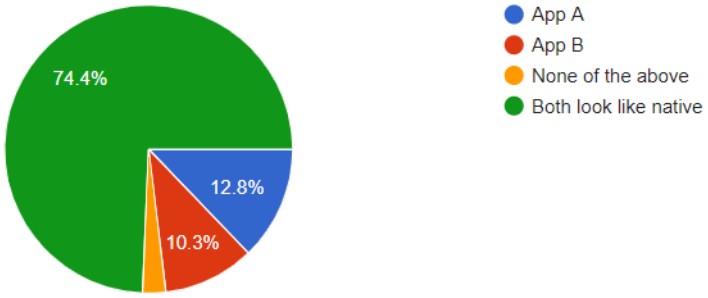


Figure 16: Diagram showing answers for looks of application A and B

If you chose anything other than "Both look like native" what differences do you see between the applications in terms of looks? describe the differences that you see
Almost no difference att all. App A has an image a little bit more centered. Other than that. Identical.
None
App A centers the image and aligned the content better.
App A is more centered on the page
smaller notifications
I tried not to think to much about it, but it was the text "Spring salad recipe" and the space between the 'Notification' that made me choose B
App B does not make use of the full screen, which leads me to believe that it is not native.
Notifications uses a larger area and easier to read
The font in App A is more towards the standard of Android Native. Also how spacing works. In native it is sometimes difficult to get teh spacing correct.
There is a slight difference in spacing both from the top to the header and to the notification list. The list looks like it has smaller font in b, but I cannot tell whats right or wrong there.

Figure 17: Individual answers for looks of application A and B

In figure 16, 74.4% answered that they thought both applications looked like native, 12.8% answered that app A looked more native, 10.3% answered that app B looked more native and 2.6% answered that none of the applications were native. The ones that answered an alternative that was not "both look like native", had to answer a follow-up question about the differences they spotted which is showed in figure 17.

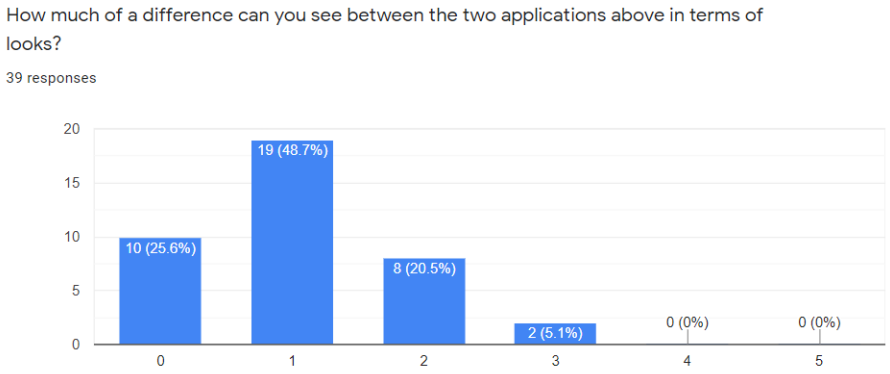


Figure 18: Answers for measured difference of application looks

Figure 18 shows a graph on the difference in looks between the two applications. The question displayed a range of 0-5 where 0 was "no difference at all" to 5 "Major difference e.g. they don't resemble at all". The majority answered around 0-2 and 2 people answered 3 which shows that the majority of people deemed the difference to be small.

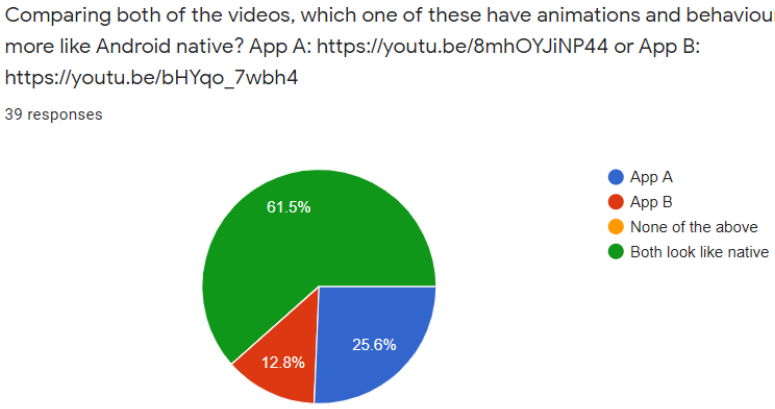


Figure 19: Answers for animations and native behaviour

Figure 19 show that the majority found both looked like native and around a quarter of the takers thought that App A behaved more native and around 12% thought that app B behaved more like native. Comparing these answers to the

first question that only regarded looks, the results show a more uncertainty of which one that behaved more native.

If you chose anything other than "Both look like native" what differences do you see between the applications in terms of animations and behaviour? describe the differences that you see
App A is more fluid and smooth in the animations.
None
Scrolling.
App B seemed to be more laggy. But that app overall was kind of blurry in the video.
det kom en lite "exlosion" när du klickar på saker
The thing at the end when scrolling is purple
The animation from reaching the top and bottom had the same color as the header on App A but had a light blue color on App B.
None
Color of end of list visual
Animation when scrolling is more android like
intensity of the color when you scroll to the top or bottom
App A has different effects for click and scrolling to the top/bottom the the list
You can see in App A how the button animation when pressed ripples out like a stone drop in the water.
The indication that you scrolled to far seemed more familiar on app B
The color of the scroll-stop is following the purple theme in A, in B it is blue.
Not sure, just "feels" right

Figure 20: Individual answers for animations and native behaviour

As seen in figure 20, there were many survey takers that felt like the behaviour of the list was different. Focus was mostly on the list animations and colour difference. One survey taker answered that the App B had more lag which corresponded with another answer that said that App A was more fluid looking.

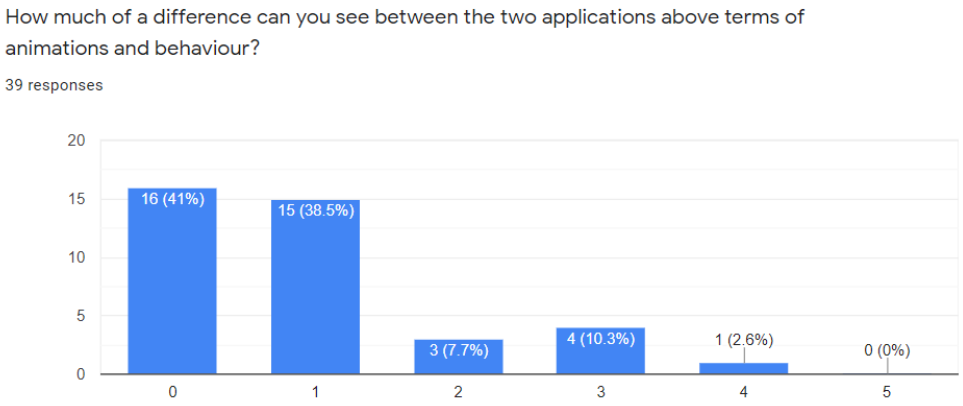


Figure 21: Answers for measure difference in animations and native behaviour

Figure 21 shows how much difference survey takers thought there was between the two applications in terms of behaviour and animations. This questions goal was to give a measurement in how much difference there can be. The majority of the answers were 0-1 of the 0-5 range that was given which were the same type of range like the question that appeared before about looks.

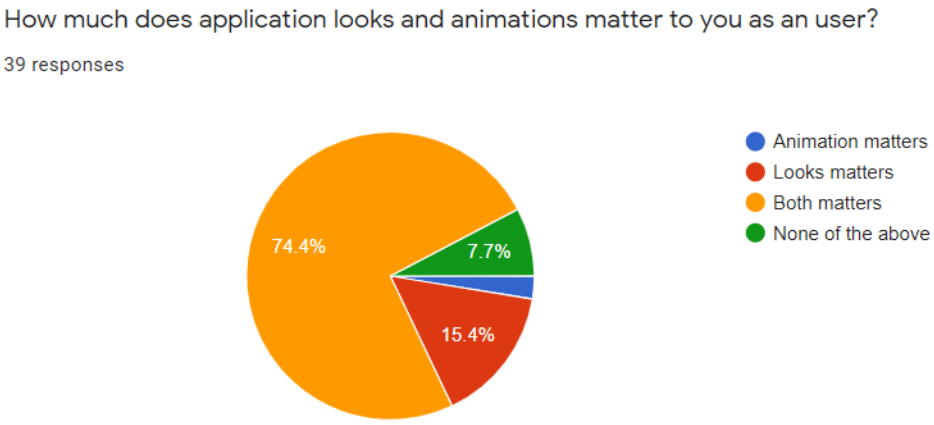


Figure 22: Answers on how much the animations, looks and behaviour matters to the user

The last question in the survey was about the importance of animations and looks for an application. In figure 22, 74% of the survey takers thought that both looks and animation behaviour mattered the most, 15% thought that looks only mattered and 7.7% did not find any of the options available important. A small portion thought that animation was most important.

8 Analysis

8.1 RQ1: How does Flutter differentiate in code size and how complex code is needed in comparison to native built applications?

These results were derived from the method that is found in 5.2, where there is a description of how the applications were developed. Collaboration with a third party developer, Consid [39], was initiated. They were to provide professionally developed applications for the experiment. This had eliminated the bias that arises with self-developed applications. The alternative was tried during the course of the work, but the collaboration with the third party failed and the method described had to be used.

The author had no prior experience in working with the three tool kits, this created an even playing field for each code base. Between each of the applications, Flutter were the easiest to work with, followed by Swift and last of all android. What made Flutter faster to work with and generated less code when having no prior experiences, was the material components. This is a library of widgets which comes ready to use and are meant to have the native look. However, the nesting that Flutter uses to mount its elements can easily turn into a mess for the developer event though there are auto generated labels that marks out the ending bracket of each widget element to make it easier to spot.

The thing that made the android native code harder to work with was how the android environment and Kotlin code handles the connection to the separate UI layout. There were many android specific parts of the code in kotlin that made the code more difficult for a beginner to handle.

iOS/Swift had a similar approach to both Flutter and android/Kotlin by connecting the UI layout as the android but having an easier styling system like Flutter.

Code complexity do not differentiate much between the three applications but has a few areas that are noticeable. This can be due to the application being small and not be comparable to if the applications were bigger. The fact that Flutter is a cross-platform and contains the code base for both iOS and android build is a fact that has to be taken in consideration.

The answer to the question based on the results would be that Flutter do keep

a shorter code base from a neutral project perspective but has a bigger project size, which means that there are a bigger amounts of revolving files that is not pure code that holds up the project. This adds up to the considerable fact that Flutter holds a structure for two builds.

There is not much more complicated code that is needed to build a Flutter application compared to a native (both operating systems), there are mostly a difference in the split and connections of the visual and the code parts which is easier in Flutter for the most part.

Code complexity was measured by the authors review. This made the RQ1 results inadequate and additional methods of measuring code complexity would be needed to draw reliable conclusions. Examples of these methods would be to have multiple people develop the same applications or measuring the code by an complexity algorithm.

8.2 RQ2: How well does Flutter perform in run time CPU usage compared to native applications on Android and iOS?

There was only one other study found on performance comparison with Flutter which was a study in an article which was said to be published but the actual paper could not be found. Beyond this paper, many others were found on comparing cross-platform tools and some native platform. The hypothesis for this question and experiment was that Flutter would perform equally or better than native. This was based on the earlier research that was made before the experiment. The testing for CPU difference in the application showed that there was not much difference in CPU usage between both native OS builds and Flutter. This can be a result of using small built applications for the experiment and the experiment could produce different results if carried out with larger applications. Flutter did not perform as good as previously thought in the hypothesis. Flutter iOS and native iOS showed a higher overall CPU usage than their android counterparts. This could have been the difference in measuring tools that was used or the tools direct measurement on application compared to overall CPU performance.

Compared to the hypothesis, the results corresponded well. The hypothesis was that Flutter would perform the same or better and the results showed that Flutter performed closely to the native builds. Comparing the results against the other studies that is found under related work, Flutter do seem to show equal CPU performance results to native, like its predecessors and competitors. The bigger differences would be located in other areas than performance.

When measuring CPU usage for the experiment, 3 measurements per application were averaged. To increase the validity of the results, more measurements could have been taken in order to eliminate the influence of any one measurement of the end results. There may exist considerable differences between the

measurements due to the human factor being involved in the experiment (i.e user input). Automating (i.e scripting) the execution of the experiment would have removed this effect.

8.3 RQ3: How much does Flutter and android native differentiate in terms of look and feel for the application users?

As shown in the results for the first question of the survey, the majority had used or had owned an android at least once. This was positive because android phones were used in the examples of the survey.

The results that revolves around looks show that even though most of the people thought that both looked like native, there was a slightly bigger percentage that thought app A looked more native over app B and a small amount that found none of the applications as native. The survey takers found no difference to a small difference between the applications. This shows that Flutter can mimic the looks of a native application without affecting the users too much. An ideal addition to the survey, would have been to use the same application in iOS to compliment the survey that only used the android applications, but this had to be omitted due to time constraints and delayed access to equipment.

Individual answers reinforced the results by motivating these differences below. Individual answers that had nothing to do with these were small issues with centering and size that could be mitigated with resources Flutter can supply, hence the other answers were sorted out.

- Not using full screen entirely
- Standard native font and native spacing

An issue that could be a problem regarding the images and videos that were used in the survey is that differences in padding between the applications are different. This could have created confusion around the amount of notification bars in the list. However, looking at the individual answers regarding looks, results show that the survey takers notice the differences in padding but do not mention the possibility of different numbers of notifications.

Looking at the other part of the survey that revolves around animation and behaviour, there were similar answers to the first question regarding looks. There was a majority that found that both behaved like native while the others chose either app A or app B. Picks for App A were however around twice the amount of as the App B pick which indicated that there was a more uncertainty with the behaviour and animation compared to the looks portion. The individual answers showed that the most noticeable differences were these:

- Smoother/fluid animations
- Touch animation on click

There were more individual answers in the animations and behaviour part that corresponded to each other, which indicates that there is a bigger importance dealing with animations and behaviour for applications. This can be related to actual using of an application versus only looking at the visuals.

In the last question, 74.4% of the survey takers found that looks, animation and behaviour mattered for an application. There were however a bigger portion that found that looks mattered more than the other alternatives. Examining the summarized answers, it suggest that an application do not always has to have the animations to compete with a user's attention, although both looks, animation and behaviour still matters to the majority of people.

9 Conclusion

Flutter is a useful toolkit that enables easy ways of creating new applications. It has gotten more and more popular recently and is talked about in the application development industry as a possible replacement of React Native and how it can be compared with native applications.

The experiment of this report revealed that there is a small difference between the CPU performance of Flutter iOS and native Swift iOS respective Flutter android and flutter native. There seem to be a difference between the performance of iOS and android but when it comes to how well Flutter can perform in CPU usage compared to native applications, there is barely a difference. The summery for this is that Flutter can perform up to par with a native application for the type of application size that was tested. To verify these results and determine that Flutter can keep up with native, further testing needs to be carried out. There are multiple other metrics that are needed in order to fully display how Flutter compares to native applications in performance.

When it came down to code size, the Flutter application had a low amount of code needed (125 lines) compared to iOS (363 lines) and android (217 lines). Flutter do not split its functionality code and visual code which lowers the amount of code and files. The Flutter code had similarities to web languages and often only required replicating to create new parts. The complexity of code that is needed for flutter, is simpler than the code that was needed for the native builds. This can however, not be fully concluded since additional methods of measuring code complexity has to be carried out for a better answer regarding the complexity and code size of all the code bases.

Appearance wise, Flutter and native seem to differentiate little to a majority of users. It is able to mimic the native looks and behaviour to a certain point. The biggest difference showed to be around the behaviour and animations of the application where the difference was noticeable on items such as list, menu, default spacing and font. Animations are however something that can be added according their documentation[40] although it requires more of the developer and application as it is not something that is accompanied by the default in Flutter.

To conclude the answers and ideas of Flutter, it is a tool with a promising feature if the community continue to grow in the direction that it is right now. The line to drawn when to choose Flutter over two separate native builds, can be chosen at the development of smaller to medium applications which are more flexible. Considering that Flutters strong side is being a cross-platform solution, Flutter still performs good on a single application base if compared to native applications. Flutter may not beat native for developing applications at this point but the results show good potential for the future although further studies needs to be done in these areas to conclude safer answers.

10 Validity Threats

During the experiment, results showed a high CPU usage for both the iOS compiled applications. This can have been caused from the way Xcode were set on measuring application performance. The readings showed more processes than the application but it was unclear if they counted as a part of the readings since only the application was chosen to be read in the settings.

Another validity threat is that there might have been insufficient knowledge when creating all the applications. This could cause the results in code complexity to be faulty and could have created differences between the applications.

A validity threat regarding the video used in the survey is that it might not be equal quality for every survey taker. This is due to the fact that the video was uploaded on YouTube and the quality is dependant on the internet speed of the user watching it. An alternative to having the youtube videos, would have been to conduct at live survey with a testing group. This was not possible due to the ongoing situation with Covid-19 and the government regulations. Another validity threat that could have affected the survey answers, is that the survey takers may have experienced that there were different amount of notification bars. This could be misinterpreted because of the difference in padding between flutter and the native applications.

As earlier stated in the analysis and conclusion, the measurement of development and code complexity is only measured by myself. The results regarding code complexity can therefore not be counted as a full reflection of the reality. Because the experiment was carried out manually, the performance measurement result may be faulty.

11 Future Work

This paper focuses on the aspects in how Flutter made applications can be used compared to a native application. A big part of Flutter is that its constantly evolves for example adding new widgets and behaviour.

The next step for studying Flutter would be to test more of its code and look into how it can be used and developed. For example, Flutter has material implemented which would add to the native look and feel of the applications. These widget features are fully able to be customized according to the Flutter developers and it would be interesting to see how far it could be taken in development. Another interesting aspect would be to test the metrics with heavier data for the applications. Consid had mentioned that they would be interested in seeing how Flutter would work in a full system, meaning that they would want to know if it could work as for example back end as well as for a web page front end.

A paper written by Amatya 2013 mentioned in the related work section, contains

a study that compares native and cross-platform applications with different loads of data, which showed that native performed better with heavier data. This is something that is not a part of this Flutter study, but can be interesting to study in the future. Studies that would include other cross-platforms for example React Native would be useful since it is Flutters competitors alongside native [10].

As mentioned in the literature review, Kotlin and Flutter have similar goals when it comes to handling platforms but different ways to achieve it. Considering this when looking at the experiment other native programming languages such as Java could be relevant to compare against Flutter too.

References

[1] O. Lifh and P. Lindholm, *Recreating a native application in react native*, 2018. DOI: diva2:1213248.

[2] A. Smith. (2019). Why businesses should start focusing on google's flutter and fuchsia, [Online]. Available: https://medium.com/swlh/why-businesses-should-start-focusing-on-googles-flutter-and-fuchsia-48e16820f2a9 (visited on 01/21/2019).

[3] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of cross-platform mobile development tools," 2012 16th International Conference on Intelligence in Next Generation Networks, Researchgate.net, 2012, p. 8.

[4] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform developement approaches for mobile applications," University of Macedonia, academia.edu, 2013, p. 7. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2490257.2490292.

[5] Google. (2019). Google flutter, [Online]. Available: https://flutter.dev/ (visited on 12/03/2019).

[6] B. D. Coninck. (2019). Flutter versus other development frameworks: A ui and performance experiment, [Online]. Available: https://blog.codemagic.io/flutter-vs-ios-android-reactnative-xamarin/ (visited on 02/06/2020).

[7] M. Bellinaso. (2018). Flutter: The good, the bad and the ugly, [Online]. Available: https://medium.com/asos-techblog/flutter-vs-react-native-for-ios-android-app-development-c41b4e038db9 (visited on 02/06/2020).

[8] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Comparing cross-platform development approaches for mobile applications," Scitpress.org, https://www.scitepress.org/Papers/2012/39045/39045.pdf: Department of Information systems, University of Münster, 2012, p. 13.

[9] L. Corral, A. Sillitti, and G. Succi, "Mobile multiplatform development: An experiment for performance analysis," Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bozen-Bolzano, Italy, diva2:664680: www.sciencedirect.com, 2011, p. 8.

[10] S. Amatya, "Cross-platform mobile development: An alternative to native mobile development," Linnéuniversitetet, Fakulteten för teknik (FTK), Institutionen för datavetenskap (DV), diva2:664680: Diva, 2013, p. 67.

[11] O. Axelsson and F. Carlström, "Evaluation targeting react native in comparison to native mobile development," Ergonomics and Aerosol Technology, LUP student papers, 2016, p. 105. [Online]. Available: https://lup.lub.lu.se/student-papers/search/publication/8886469.

[12] M. R.-S. Guerra, "Cross-platform development frameworks for the development of hybrid mobile applications: Implementations and comparative analysis," ESCUELA SUPERIOR DE INGENIERÍA GRADO EN INGENIERÍA INFORMÁTICA, hdl, 2018, p. 86. [Online]. Available: http://hdl.handle.net/10498/20951.

[13] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein, *Survey, comparison and evaluation of cross-platform-mobile-application-development-tools*, 2013.

[14] Google. (2019). Google flutter sdk releases, [Online]. Available: https://flutter.dev/docs/development/tools/sdk/releases (visited on 01/21/2019).

[15] G. Flutter. (2020). Flutter setup, [Online]. Available: https://flutter.dev/docs/get-started/install (visited on 04/30/2020).

[16] Android. (2020). Android studio setup, [Online]. Available: https://developer.android.com/studio (visited on 04/30/2020).

[17] A. Inc. (2020). Xcode setup, [Online]. Available: https://developer.apple.com/xcode/ (visited on 04/30/2020).

[18] W. Danielsson, "React native application developement," Diva, diva2:998793: Faculty of Computer science LIU, 2016, p. 70.

[19] M. E. Joorabchi, A. Mesbah, and P. Krunchten, "Real challenges in mobile app development," University of British Columbia, University of British Columbia, 2013, p. 10. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.724.2463&rep=rep1&type=pdf.

[20] B. Armour. (2018). 5 key benefits of native mobile app development, [Online]. Available: https://clearbridgemobile.com/benefits-of-native-mobile-app-development/ (visited on 05/03/2020).

[21] C. Griffith. (2019). What is cross-platform app development, [Online]. Available: https://ionicframework.com/resources/articles/what-is-cross-platform-app-development (visited on 02/13/2020).

[22] M. Bellinaso. (2018). Flutter vs react native for ios development, [Online]. Available: https://medium.com/asos-techblog/flutter-vs-react-

native-for-ios-android-app-development-c41b4e038db9 (visited on 05/07/2020).

[23] C. Software. (2019). What is flutter here is everything you need to know, [Online]. Available: https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f (visited on 05/07/2020).

[24] G. Flutter. (2020). Faq flutter, [Online]. Available: https://flutter.dev/docs/resources/faq (visited on 02/13/2020).

[25] Google. (2020). Dart, [Online]. Available: https://dart.dev/ (visited on 06/05/2020).

[26] M. Satei, "Ott video-oriented mobile applications development using cross-platform ui frameworks," KTH Royal Institute of Technology, School of eletrical engineering and computer science, diva2:1343759: Diva, 2018, p. 90.

[27] C. Software. (2019). What is flutter? here is everything you should know, [Online]. Available: https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f (visited on 03/05/2020).

[28] Google. (2020). Material io components, [Online]. Available: https://material.io/components (visited on 05/03/2020).

[29] Stackoverflow. (2019). Stackoverflow survey 2019, [Online]. Available: https://insights.stackoverflow.com/survey/2019 (visited on 06/01/2020).

[30] ——, (2020). Stackoverflow survey 2020, [Online]. Available: https://insights.stackoverflow.com/survey/2020 (visited on 06/01/2020).

[31] Jetbrains. (2020). Kotlin faq: What is kotlin? [Online]. Available: https://kotlinlang.org/docs/reference/faq.html (visited on 02/26/2020).

[32] M. Bhatnagar. (2018). Statically typed languages vs dynamically typed languages, [Online]. Available: https://android.jlelse.eu/magic-lies-here-statically-typed-vs-dynamically-typed-languages-d151c7f95e2b (visited on 02/26/2020).

[33] Jetbrains. (2020). Kotlin-faq, [Online]. Available: https://kotlinlang.org/docs/reference/faq.html (visited on 03/11/2020).

[34] G. Developers. (2020). Android layout, [Online]. Available: https://developer.android.com/guide/topics/ui/declaring-layout#kotlin (visited on 05/07/2020).

[35] A. Inc. (2020). Swift, [Online]. Available: https://swift.org/about/ (visited on 05/07/2020).

[36] L. organisation. (2020). Llvm infrastructure, [Online]. Available: https://llvm.org/ (visited on 05/06/2020).

[37] A. Inc. (2020). Swift compiling, [Online]. Available: https://swift.org/swift-compiler/#compiler-architecture (visited on 05/06/2020).

[38] ——, (2020). Xcode interface builder, [Online]. Available: https://developer.apple.com/xcode/interface-builder/ (visited on 05/07/2020).

[39] C. Karlskrona. (2020). Consid karlskrona, [Online]. Available: https://consid.se/en/kontor/karlskrona-2/ (visited on 06/03/2020).

[40] G. Flutter. (2020). Google flutter animations, [Online]. Available: https://flutter.dev/docs/development/ui/animations (visited on 03/12/2020).

12 Appendix

12.1 Appendix A - Device Specifications

12.1.1 Device specifications

Operating System	Chipset	CPU	GPU	Memory
Android 8.0.0 (Oreo)	Exynos 8890 Octa (14 nm)	Octa-core (4x2.3 GHz Mongoose 4x1.6 GHz Cortex-A53)	Mali-T880 MP12	4 GB RAM
iOS 13.4	Apple A10 Fusion (16 nm)	Quad-core (2.34 GHz 2x Hurricane + 2x Zephyr)	PowerVR Series7XT Plus (six-core graphics)	3 GB RAM

12.1.2 Tools and packages

- Xcode 11
  - SwiftUI version that came with Xcode 11
- Android studio 3.6.1
  - Kotlin 1.3.70
- Flutter 1.12.13+hotfix.8
  - dart 2.7.0
  - font awesome flutter: 8.0.1
  - cupertino icons: 0.1.2
  - hexcolor: 1.0.1

12.1.3 Code base

Code base for the three applications classes can be downloaded here:
`https://drive.google.com/file/d/1sFCSRbfbmi9nG40jzieSSxpn_j8QpjcG/view?usp=sharing`

12.2 Appendix B - Release years of mobile development tools

Language/Tool	Release Year
Xamarin	2014
Kotlin	2016
Swift	2014
React Native	2015
Flutter	2018
PhoneGap	2011

12.3 Appendix C - Survey details and survey questions

Link to spreadsheet: full answer spreadsheet

1. How familiar are you with Android smartphones? (1-5)
2. Which one of the pictures below looks more like an Android native application? magnified image on link: https://imgur.com/a/Exb594z (App A, App B, None of the above, Both look like native)
3. If you chose anything other than "Both look like native" what differences do you see between the applications in terms of looks? describe the differences that you see? (Long-answer text)
4. How much of a difference can you see between the two applications above in terms of looks? (1-5)
5. Comparing both of the videos, which one of these have animations and behaviour that look more like Android native? App A: https://youtu.be/8mhOYJiNP44 or App B: https://youtu.be/bHYqo_7wbh4? (App A, App B, None of the above, Both look like native)
6. If you chose anything other than "Both look like native" what differences do you see between the applications in terms of animations and behaviour? describe the differences that you see? (Long-answer text)
7. How much of a difference can you see between the two applications above terms of animations and behaviour? (1-5)
8. How much does application looks and animations matter to you as an user? (Animation matters, Looks matters, Both matters, None of the above)

12.4 Appendix D - Detailed measurements for each environment version

Measure(Flutter iOS)	Highest	Lowest	mean	Standard Deviation	set of values (Sample values)
M1	89.6%	19.0%	33.0%	18.39126604	68.7, 20.5, 26.7, 28.4, 34.9, 19
M2	115.0%	19.9%	40.0%	25.52022074	85.1, 26, 30.3, 22.7, 55.9, 20.5
M3	100.6%	17.5%	32.9%	10.10893994	17.5, 45.6, 26.1, 32.1, 40.3, 36.2
Average	101.7%	18.8%	35.3%	18.00680891	

Measure(Flutter Android)	Highest	Lowest	mean	Standard Deviation	set of values (Sample values)
M1	31.0%	1.0%	13.8%	7.80811544	0,10, 17,18,16,22
M2	26.0%	1.0%	10.5%	9.81325634	0, 6, 27, 3, 15, 12
M3	40.0%	1.0%	15.5%	10.25182911	0,10,16,17,19,31
Average	32.3%	1.0%	13.2%	9.29106696	

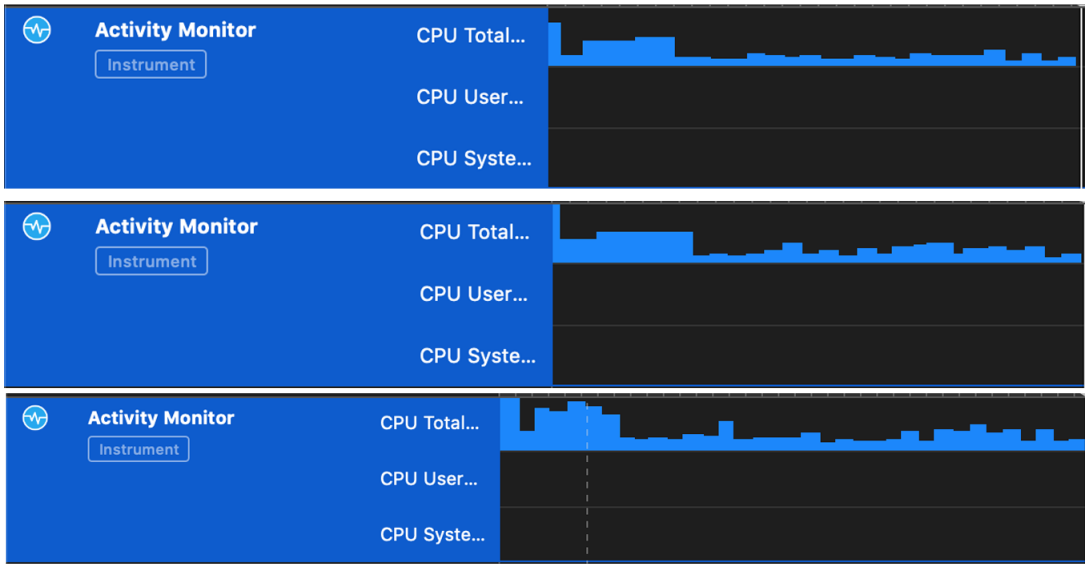
Measure(Native iOS)	Highest	Lowest	mean	Standard Deviation	set of values (Sample values)
M1	94.0%	18.9%	26.8%	12.13667994	51, 26, 19.4, 22.3,23.7, 18.7
M2	91.6%	12.6%	32.6%	13.42708705	33.2, 42.1, 12.6, 23.9, 50.9, 33.1
M3	92.7%	11.5%	39.3%	15.69705917	54.1, 29.9, 32.7, 26, 64, 29.6
Average	92.7%	14.3%	32.9%	13.75360872	

Measure(Flutter iOS)	Highest	Lowest	mean	Standard Deviation	set of values (Sample values)
M1	89.6%	19.0%	33.0%	18.39126604	68.7, 20.5, 26.7, 28.4, 34.9, 19
M2	115.0%	19.9%	40.0%	25.52022074	85.1, 26, 30.3, 22.7, 55.9, 20.5
M3	100.6%	17.5%	32.9%	10.10893994	17.5, 45.6, 26.1, 32.1, 40.3, 36.2
Average	101.7%	18.8%	35.3%	18.00680891	

12.4.1 Android Flutter graphs

12.4.2 Android native graphs

12.4.3 iOS flutter graphs



12.4.4 iOS native graphs