# Flutter and iOS Application Comparison

**An Empirical Metric Analysis of Performance and User Experience**

## Bachelor Thesis


| | |
|---|---|
| submitted by: | Philip Krück |
| Date of Birth: | 04.11.1998 |
| Matriculation Number: | 3938 |
| Company Supervisor: | Jan Jelschen |
| First Reviewer: | Dr. Oliver Becker |
| Word Count: | < 12.000 (text + footnotes) |
| Degree Program: | B.Sc. Business Informatics (A Track 2018) |
| University: | Hamburg School of Business Administration |
| Submission Date: | 09.04.2021 |
| Partner Company: | apploft GmbH |



HSBA HAMBURG SCHOOL OF BUSINESS ADMINISTRATION

# Abstract

To be written... (max: 300 words)

# Contents

# Chapter 1

# Introduction

Mobile platforms are dominated by two players - Apple and Google with their respective operating systems iOS and Android. Cumulatively, they form a duopoly in the smartphone operating systems market with a combined usage share of 15.2% for iOS and 84.8% for Android in 2020 according to to the International Data Corporation (**IDC2021**).
To develop a mobile software application (*app*) for both target platforms, the corresponding development environments and technologies are utilized for each platform. This leads to a doubling of cost, development time, and the need for knowledge of two different application development paradigms. As a result, cross-platform frameworks such as Xamarin (**Xamarin2021**), React Native (**Facebook2021**), and Ionic (**Ionic2021**) have been created.

The fundamental principle behind these frameworks is the provision of a unified tech stack operating on a single code base leading to increased development speed while also providing the ability to deploy for both mobile operating systems.

Generally, cross-platform frameworks utilize webviews[1] or a software bridge to communicate with the underlying host platform plugging into native interfaces. In both cases, communication channel delays may occur during app runtime leading to reduced execution speed. In addition, cross-platform frameworks deliver abstract interfaces over multiple native interfaces leading to a decreased subset of functionality and especially impaired UI customizability. These inherent architecture attributes (further see Section 2.1) explain both impeded performance (**Ebone2018** and **Corbalan2019**) and user experience (**Mercado2016** and **Angulo2014**) compared to native technologies.

However, over the past 2 years one particular cross-platform technology with certain dis-

1. Webviews are UI component in both iOS and Android to display web content such as *HTML*, *CSS* and *JavaScript*

tinct attributes has risen strongly in popularity (**Statista2021**): Flutter (**FlutterDev20**) is a newly developed open-source UI toolkit by Google. It has a nonconventional approach to cross-platform development in that every app ships with the framework's rendering engine (Section ...). Thereby, Flutter bypasses host platform communication by avoiding the underlying system app SDK for terms of UI rendering. Flutter apps are compiled in native binary format which can be directly executed by the mobile computing device (see Section 2.2). Furthermore, the framework provides a *"[...] collection of visual, structural, platform, and interactive widgets"* (**GoogleWidgets2021**) for UI customization. It seems that Flutter addresses the exact same issues that are generally criticized about cross-platform solutions.

## 1.1 Motivation

As a digital agency specialized on native iOS and Android development, **apploft GmbH** (the partnering company of this thesis) (**apploft2021**) is highly interested in Flutter. The implications of using this framework could be wide-ranging, including the extension of the services portfolio to clients with lower budgets. For example, startups which are unsure about product/market fit (**Andreesen2007**) and held by tight budget constraints are especially keen on reaching the maximum number of potential customers with their app. Flutter could be utilized for a fast iteration of a product deployable to both mobile platforms. Worth noting are the benefits of serving smaller customers like startups which include higher growth potential for long-term cooperation and risk diversification in apploft's client portfolio.

The possible updside of implementing Flutter exceeds the acquisition of small clients. Instead of focusing on platform customization with native tooling, more effort could be directed into developing unique custom features. Furthermore, infrastructure setup, package development and app updates would only be necessary for one codebase.

Since the aforementioned economic incentives aren't necessarily company-specific, they are relevant for mobile application developers at large. Scientifically, this thesis may be the basis for future work on Flutter's architecture attributes and their contribution to runtime performance and UI rendering of frontend toolkits, in general.

## 1.2 Problem Statement

To the best of the author's knowledge, there are no peer reviewed articles comparing the performance or usability to native apps[2] since Flutter was first released in March 2018 (**FlutterReleases2020**), This leaves an especially interesting gap in the literature, since both aspects are the topmost perceived challenges of cross-platform frameworks considered from an industry-perspective (**BioernHansen2019**).

## 1.3 Thesis Objective

This thesis will focus on comparing Flutter's framework technology to iOS specifically. The assumption made in this paper, is that mimicking Android-specific appearance and behavior should be rather straightforward as both Flutter and Android utilize Google's **Material design** (**Google2021**) for their default components. Additionally, testing the framework against iOS is especially interesting as it seems less likely that Flutter would be able exploit system specific properties for performance optimization in Apple's closed operating system.
The aim of this thesis is derived based on the initially stated problems with cross-platform frameworks and the current lack of research on Flutter's lofty marketing claims to solve those drawbacks. Specifically, Google's assertion that Flutter can match *"native performance"* and the framework can be utilized for building *"expressive and flexible UI"* (**FlutterDev20**) will both serve as inductively derived hypotheses that shall be empirically verified or falsified individually by this thesis:

$H_P$: The Flutter framework yields comparable **performance** to native iOS app development frameworks for iPhone.

$H_U$: Comparable **user experiences** can be created with Flutter and native iOS frameworks for iPhone.

## 1.4 Methods

An architypal native mobile app has been chosen as a case study for the evaluation of both hypotheses $H_P$ and $H_U$. Based on typical mobile application features (explained in detail in Section 3.1), *Kickdown* (**Kickdown2021**) - an online car auction app - was chosen for this thesis. The app has already been developed for iOS by apploft and released to the App Store

2. A search for relevant articles has been conducted using Google Scholar, Sci-hub and IEEE Xplore.

in February of 2021 (link to appstore). For the purposes of comparison, a Flutter equivalent has been developed mimicking the relevant subset of UI and functionality of the original app (see Section 3.1). Both the original and Flutter replica app are used for subsequent hypothesis testing.

### 1.4.1 Performance comparison

The assessment of the performance hypothesis $H_P$ has been conducted by profiling specific performance metrics including CPU, GPU and memory usage for particular use case flows in the original and Flutter application. The profiling results have been analyzed using common statistical techniques. A further explaination of the measurement process is detailled in Section 3.2.

### 1.4.2 Usability comparison

The analysis of the usability hypothesis $H_U$ has been evaluated by the means of semi-structured expert interviews. Specifically, study participants have been be asked to evaluate both the Kickdown iOS and Flutter application clone along various metrics.

A detailed explanation of the interview process is given in Section 3.3.

## 1.5 Scope & Limitations

The feature set of the implemented app is representative for most, but not every type of app (see Section 3.1). Therefore the results cannot be generalized to every type of possible app. However, they should be seen as indicators of Flutters value as a cross-platform framework for the archetypal mobile app. Furthermore, the deductively chosen methodology yields the potential of finding adjacent hypothesis which may be further explored by other researchers. Additionally, the usability study doesn't provide statistical significance due to its qualitative nature. Nevertheless, the depth of detail in expert interviews is much greater compared to quantitative methods, and unthought of considerations may be suggested by the interviewees. This research attribute is especially compelling given the current research state on the Flutter framework as mentioned above.

## 1.6 Thesis Structure

*- TODO: Describe structure of thesis and summarize each chapter. Do this once the other chapters are actually written.*

# Chapter 2

# Flutter

consider naming it theoretical background

- introduce native technology as a baseline

- create graphic with over of mobile development approaches

- revisit sections for PWA, Hybrid apps, runtime-based and interpreted apps of Springer article

or name it overview of cross-platform development approaches

## 2.1 Mobile Developement Tiers

### 2.1.1 Web Apps and Progressive Web Apps

Web apps are applications run and rendered in the browser. The underlying technologis are HTML, CSS and JavaScript with popular frameworks used for the development process such as Angualar (ref), React.js (ref) and Vue.js (ref). Web apps rely on the Browser and an internet connection. They do not have direct access to hardware capabilites such as the camera or the file system. Progressive Web Apps (PWAs) function similarly to web apps but provide additional capabilities such as offline use, locally cached data, push notifications, and the ability to add them to the home screen.

### 2.1.2 Hybrid Apps

Hybrid Apps utilize the same technologies as web apps, but they are rendered in a platform web view. Additionally they provide the ability to interact with native APIs through a platform bridge (see 2.1). Popular framework choices for building Hybrid Apps include Ionic and Cordova

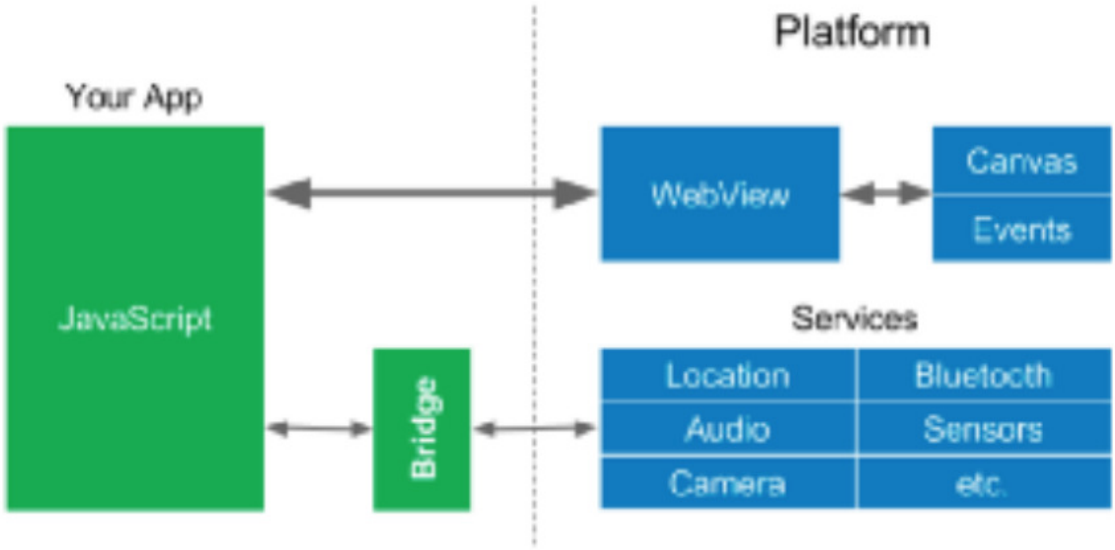


Figure 2.1: Hybrid architecture (image to be replaced).

Hybrid apps unlike web apps are shippable to official stores.

(ref: https://www.freecodecamp.org/news/a-deeply-detailed-but-never- definitive-guide-to-mobile-development-architecture-6b01ce3b1528)

### 2.1.3 Web Native Apps

Web Native Apps which are built with frameworks such as React Native (ref) or Native Script (ref) utilize the OEM components instead of web views. This is achieved by using JavaScript calls to a platform bridge which transpiles the JavaScript code into native platform code (see 2.2). The views are thus in Web Native Apps rendered with native technologies.

### 2.1.4 Cross Compiled Apps

Cross Compiled Apps utilize a compiled programming language such as C# (used for Xamarin) to generate byte or machine code which is then executed on the host platform. This machine code performs similar actions as web native apps in that platform APIs and OEM widgets are called (see 2.2). Flutter can also be classified as a cross compiled architecture. However, Flutter ships its own widgets and rendering engine instead of relying on the host platform (see 2.3). These technicalities will be explained in Section 2.2.

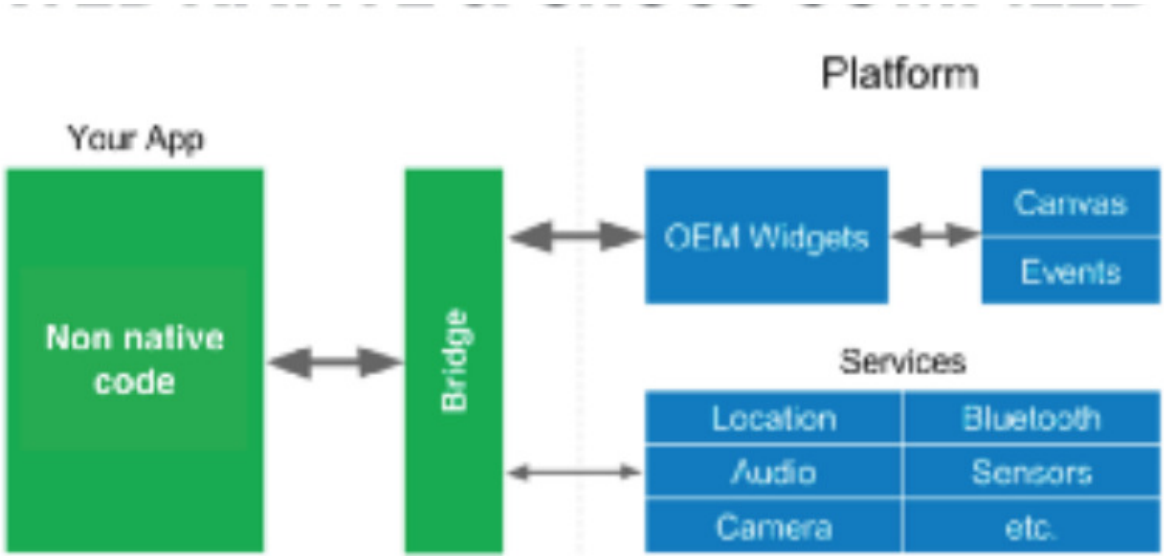


Figure 2.2: Web Native and Cross Compiled Architecture (image to be replaced).

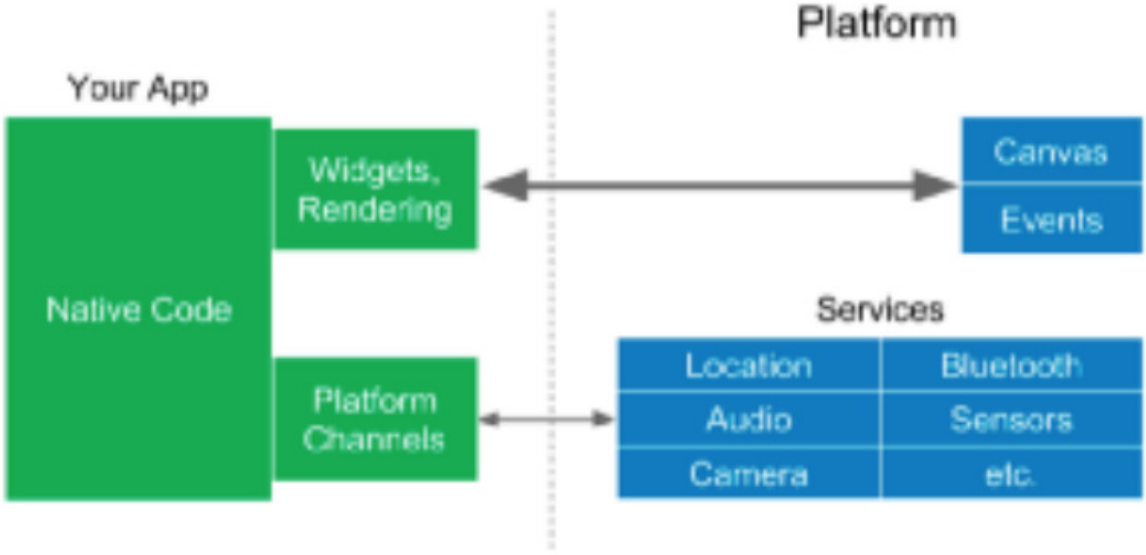


Figure 2.3: Flutter Architecture (image to be replaced).

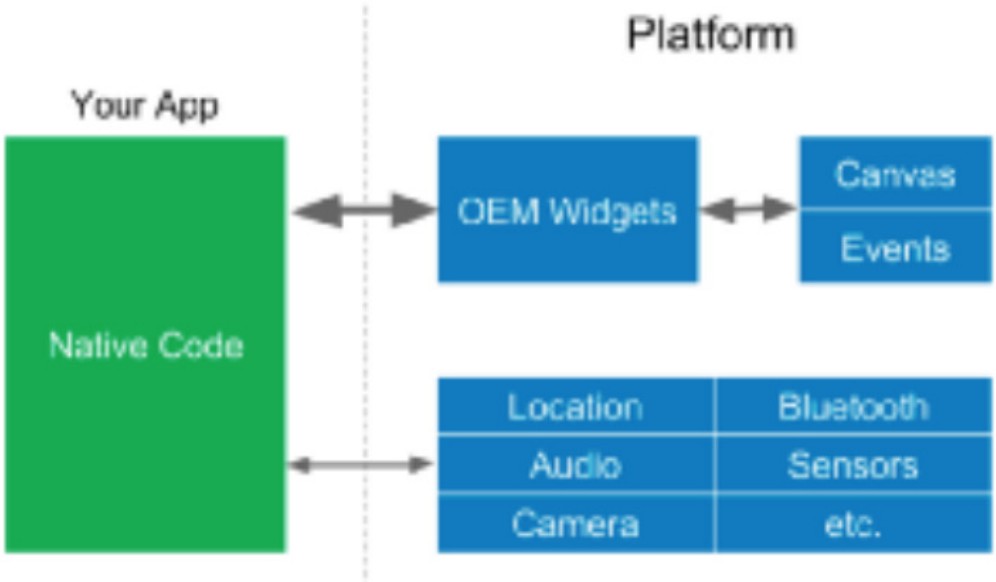


Figure 2.4: Native Mobile Architecture (image to be replaced).

### 2.1.5 Native Mobile Applications

Native applications directly communicate with the OEM components and APIs which are part of the operating system and thus tightly integrated into the platform. Making full use of hardware and software integration, native apps yield the highest performance and user experience.

## 2.2 Flutter Framework Architecture

Flutter's framework architecture is composed of three distinct layers: framework, engine and embedder. The framework acts as the top most layer which app developers interact with. It features the widgets, animation and gestures API as well as platform specific UI components delivered through the Material (Android) and Cupertino (iOS) package.

Below the framework layer lies the engine which unlike the framework isn't written in Dart, but in C and C++. Presumably, this design choice has been made to easily allow the production of native binaries. This act of cross-compilation is Flutter's technical value proposition to increase execution performance as stated in Section 1. The engine layer includes Skia - a 2D graphics rendering engine which is also used in the Chrome web browser.

At the bottom lies the embedder layer. Its sole purpose is to integrate the Flutter application into the platform-specific environment by providing native plugins and event loop interoperation. Additionally, app package formatting is provided in the same way as for native

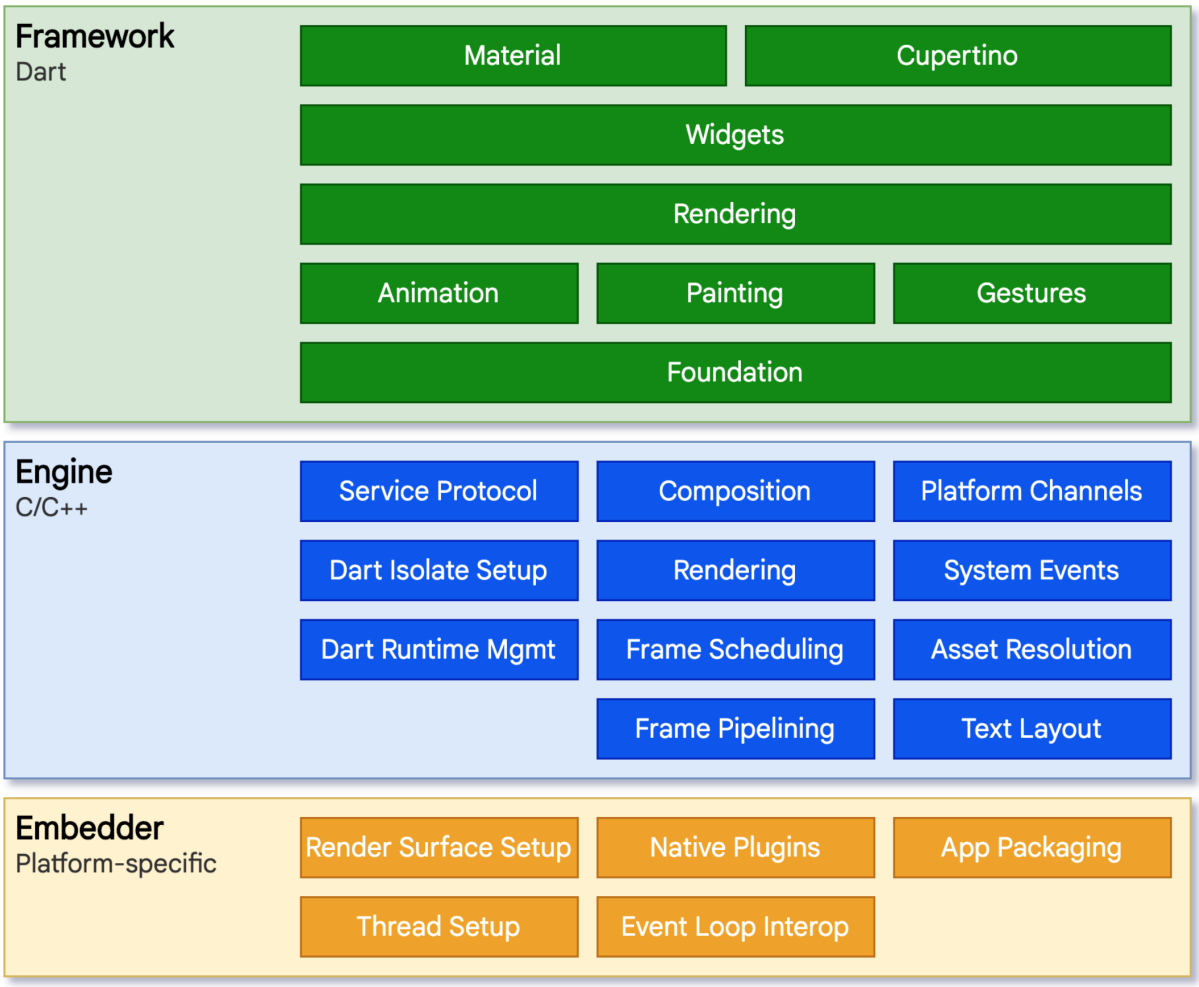


Figure 2.5: Flutter's Layered Architecture

apps. The host operating is thus not able to differentiate between a Flutter and a natively written app. Flutter aims to minimize application size as it's already shipping the entire framework including the rendering engine with each app which amounts to approximately 4.5 mb (https://flutter.dev/docs/resources/faq). Common app platform plugin functionality like camera access or webview interaction are extracted in separate packages. Networking, animations or other platform agnostic functionality is also bundled in packages.

(ref to include https://flutter.dev/docs/resources/architectural-overview).

### 2.2.1 Programming Language and Compilation

Flutter apps are written in Dart an multiparadigm programming language syntactically similar to Java. (https://dart.dev/overview#language)

During Flutter development, apps run in the Dart Virtual Machine with JIT compilation. This offers stateful hot reload which allows reload changes without needing to fully recompile

the app leading faster development iteration. For release purposes, Flutter applications are Ahead-of-Time compiled into native machine code including the Intel x86 and ARM instruction sets. (https://flutter.dev/docs/resources/architectural-overview).

### 2.2.2 Rendering and UI State

Flutter apps are fundamentally composed of widgets which declare structural, stylistic and layout elements for building the user interface. Each widget may have 0, 1 or multiple children which in turn create a tree structure of parent-child relationships. For example, a Column which has the purpose of laying out its children vertically has both a Text and an Image as its children. Both the Text, and Image widgets have no children and represent the leaf nodes in the tree. The Column is an invisible structural layout element while Text and Image are stylistic. [1] The entry point of every Flutter app is either a 'MaterialApp' or 'CupertinoApp' widget which also marks the root of the tree. Based on this tree structure Flutter can determine where and how elements should be drawn on screen, and instruct its graphics engine accordingly. This concept alone is not yet sufficient to enable modern mobile applications with complex UI changes or animations based on asynchronous events like user interaction. Widgets are mappings from state to a UI representation. When the state changes during runtime Flutter creates a new widget tree, diffs it against the old one and then redraws only its changes to the screen. This declarative approach simplifies UI development in the sense that the developer does not need to keep track of UI state which can grow exponentially with the increase of UI components on screen.

### 2.2.3 Method Channels

To utilize platform functionality like camera access, geolocation or other sensor data, Flutter communicates with the platform's native APIs via method channels (see architecture diagram). Common functionality is already provided by Flutter and third party packages, but custom platform channel functionality may be implemented as required. (reference Flutter doucmentation...)

## 2.3 Flutter's Limitations

Unlike other cross-platform frameworks, Flutter doesn't use OEM components, but its own components instead (widgets). When the underlying OEM component of the host platform changes

1. Technically both Text and Image do have an implicit single child widget which is created by the framework.

with an OS update, Flutters' framework needs to updated to resemble this new functionality. Flutter always has to continuously replicate these changes in their framework layer. As Google is using Flutter itself in multiple production apps (cite something), the company has an incentive to swiftly replicate OS features. Additionally, since Google also owns Android, coordination may be easier and not as surprising. The benefit though is that once the new widget is rebuilt in Flutter it can even be shipped to older operating systems.

# Chapter 3

# Methodology and Study Design

This chapter explores the employed methodology for testing both the performance hypothesis $H_P$ and the usability hypothesis $H_U$ (mentioned in Section 1.3). The goal is to explicitly show the reasoning for the chosen methods as well as provide specific method implementation details to aid transparancy about the obtained results discussed in Chapter 5.
Generally, the employed methodology to achieve the research aim (Section 1.3) is a replication of a relevant subset of functionality of an existing iOS app using Flutter. Thereby, the original app acts as a baseline with which the Flutter clone can be comparatively evaluated. Based in this comparison, the research question - whether the Flutter framework can match native performance and provide equivalent usability (Section 1.3) - is answered. Instead of creating artificial use cases, taking advantage of an existing app provides realistic instances for performance as well as user interface testing.
The first section in this chapter (Section 3.1) details the decision process for selecting the baseline testing app as well as its feature reduction for further comparison. The subsequent two sections (Sections 3.2 and 3.3) explore the specific methods and reasoning for the performance and usability comparison respectively.

## 3.1 Baseline App Testing Decision Process

The procedure for choosing the case study app is based on a filtering process of 4 steps (i.e. opposed constraints):

1. The app is built and maintained by apploft.

2. The app includes common application **features**.

3. The app uses modern iOS framework technologies.

4. The app conforms to the human interface guidelines (HIG) by Apple (**Apple2021a**).

The reasoning behind selecting the above filtering constraints is detailed in the following paragraphs.

**1. Creation and Maintainance by apploft**

This constraint was opposed on the filtering process such that both a contact person (apploft employee) is available for code specific questions. In conjuction with access to the original source code as a reference, functionality may be implemented similarly in the Flutter replica as to facilitate comparability with the baseline app. E.g. a particular algorithm could be implemented homogeneously in the Flutter application to produce equal runtime complexities and thus be retracted as a confounding variable explaining performance differences. Furthermore, access to the original source code provides the ability to reduce unnecessary features of the original app to ensure equivalent comparisons against the Flutter clone.

**2. Inclusion of Common Application Features**

On the one hand, including only common features constrains the tested functionality to fit within the scope of this thesis. On the other hand, it allows for the ability to generalize the results to a majority of iOS applications[1].

**3. Use of Modern iOS Frameworks**

Constraining the baseline app to be built with modern iterations of iOS framework technology ensures a fair comparison against the replica app built with the recently released Flutter framework.

**4. Conformance to HIG**

Conforming to Apple's HIG ensures the original app looks native to the iOS platform. Building a matching Flutter clone is especially interesting regarding the usability hypothesis $H_U$ as it would stretch the realm of possibility for Google's framework. In addition, providing a recognizable UX for iOS users would keep participants in the usability study (detailed in Section 3.3) focused on noticing differences instead of being distracted by an ambiguous UI.

1. This assumes that the observed apps for selecting features are indicative of the archetypal iOS app.

Based on the above constraints, a small study was conducted looking at 15 apps developed by apploft (constraint 1) from 9 different iOS App Store categories. The goal is to find common application features (constraint 2) among them. As for the purposes of this study, a **feature** is defined as either

(a) a generalizable UI component which is non-trivial, or

(b) an underlying technical attribute influencing the user experience.

Trivial UI components (a) such as buttons or text weren't considered **features** as they are omnipresent throughout every app. As for (b), a technical attribute has to influence the user experience to be incorporated as the purpose of this thesis is testing Flutter's value as a UI framework (see Section 1.3). For example, networking can be viewed as a **feature** if fetched data is displayed via the UI, but is not a **feature** if the sole purpose of networking within an app is to extract analytics data.

Furthermore, a **feature** had to appear at least twice before being added as a result (see Table [initial table]).
Continuing the filtering process, as per constraint 2 uncommon features - features appearing in less than 50% of observed apps - are excluded. This reduces the list of features to the following:

- Networking
- Login/Authentication
- Tab navigation[2]
- Stack navigation[3]
- Keyboard interaction
- Vertically scrolling collections[4]
- Horizontally scrolling collections[5]

2. ...
3. ...
4. ...
5. ...

- Webview component[6] integration

Out of the 15 initially tested apps, 5 include all of the above **features** (conforming to constraint 1 and 2) (see Table [table after applying constraint 2]). Kickdown (see Section ...) is chosen among the remaining contestants for the baseline testing app. It was most recently released (Feb 2021) and is therefore built with modern iOS technologies (constraint 3) and complies to the most recent iteration of the **Human Interface Guidelines** (constraint 4).

The login and signup mechanism - although a common **feature** - is removed from the original app for baseline testing. This is due to the fact that textfield and button interaction as well as networking is already present in other parts of the app and would yield no further insight regarding the hypotheses evaluation.
The Flutter app is implemented as closely as possible to the original application to avoid an asymmetrical comparison as detailed in Section 4.

## 3.2 Performance Comparison

The methodology chosen to test the performance hypthesis HP (Section 1.3) is a quantitative measurement of computational resources during app runtime. Measurements are performed for specific load conditions (i.e use cases). In the process, the original app acts as an empirical baseline for testing the Flutter replica against.
Directly benchmarking system resources provides insight whether the Flutter framework consumes compute resources efficiently under typically imposed load settings. Furthermore, system benchmarking metrics are the underlying cause of more ephemeral measures for testing the system load itself, e.g. page load time. In addition, the chosen compute resources (explained in Subsection 3.2.1) are easily measured using software tooling (Subsection ...) which aids the traceability as well as reproducibility of this particular study methodology.

### 3.2.1 Selected Performance Measurement Variables

The following paragraphs introduce the selected performance measurement variables. Concretely, a brief definition is given as well as the reasoning for including the particular metric in this study with regards to evalutating the performance hypothesis (Section 1.3).

6. ...

#### CPU Utilization

CPU utilization is defined as the CPU time (**FSF1988**) of a task divided by its overall capacity expressed as a percentage. The CPU as well as its integrated GPU[7] are responsible for graphics rendering related computations. Generally, high CPU usage is an indicator of insufficient processing power of the executed task. Therefore, testing CPU utilization directly assesses whether Flutter's framework processing requirements for UI rendering can be fulfilled given the testing hardware (see Subsection 3.2.2).

#### Frames per Second

Frames per Second (FPS) describes the rate at which the system, and in particular, the GPU completes rendering individual frames. The FPS rate directly determines the smoothness of UI animations and transitions (**Google2020**).

#### Memory Utilization

Memory utilization is the percentage of available memory capacity used for a specific task. A High level of memory usage "[...] affects the performance of actual running tasks, as well as interactive responsiveness" (**Ljubuncic2015**).

### 3.2.2 Measurment Process

The measurement process for the individual metrics is further split into specific user actions which are executed and tested on both the iOS and Flutter app separately. These were chosen to test all relevant facets of the app (see Section 4.1) and ensure a necessary load on the system:

- **app start:** The app is freshly installed on the test device, opened and idle until the visible postings are loaded.
- **scrolling:** On the postings overview screen, the posting cards are scrolled fully to the bottom and subsequently back to the beginning.
- **detail view:** From the postings overview, the first posting is tapped to navigate to the detail view. Afterwards the back button is tapped to navigate back to the overview.
- **image gallery:** The image gallery of a posting is opened from the detail view of a posting and the first 10 images are viewed by swiping.

7. The GPU is an integrated chip within the System-on-a-Chip (SoC) architecture (**Martin2001**) for all iPhone processing units (**WikiChip2020**).

For each **user action**, the average of all values over time is recorded. This process is then repeated 3 times and averaged. The exact number of experiment repitions was chosen as a tradeoff between marginal accuracy increase and additional experiment execution time.
Furthermore, 2 testing rounds are devised on separate devices. The iPhone 12 and iPhone SE are chosen as the upper and lower bounds of hardware performance respectively. The lower bound is defined in this case as per Apples recommendation to set the deployment target to the current operating system version (iOS 14 at time of writing) minus one (iOS 13) which lists the iPhone SE as the oldest supported device (**Apple2021b**). iOS 13 is also the deployment target of the original Kickdown app.
To reduce measurement bias, the device is restarted before each measurement to ensure that all irrelevant background processes are cancelled.

### 3.2.3 Profiling Tools

**Xcode Instruments** (**Apple2019**) - a part of the **Xcode** IDE tool set - are used for profiling the individual metrics. It provides multiple preconfigured profiling trace instruments. For the purposes of this thesis, the **Time Profiler** tool (see Figure ...) is used for CPU (ref ...), the **Core Animation** tool (see Figure ...) for FPS (ref ...), and **Allocations** tool (see Figure ...) for memory usage (ref ...) quantification (see ) over time.

### 3.2.4 Evaluation Process

To better understand the data gathered, it is subsequently examined using exploratory data analysis (EDA) (**Tukey1977**). *To be continued a bit...*

## 3.3 User Experience Comparison

This Section explores the usability hypothesis evaluation methodology. Specifically, laying out the procedure to answer the question of whether or not the Flutter framework is capable of reproducing native iOS application user experiences (see Section 1.3).

Generally, as UX is built for other humans, any evaluation is prone to subjectivity and perception biases (**Tversky1974**). Therefore, it is difficult to capture these impressions quantitatively in a reproducable manner.
However, the user experience of an app is directly dependant upon sufficiently underlying per-

formance (e.g. for scrolling fluidity) and therefore the results of the performance comparison (Section 5.1) form the basis of the evaluation of the usability hypothesis $H_U$. Utilizing a mixed approach as a study methodology combining both the quantitative performance comparison and a qualitative method will draw upon the strengths of both approaches.

Specifically, semi structured interviews with subject matter experts (Cf. Vgl. Liebold/Trinczek 2009, S. 32 ff.) are conducted to evaluate the baseline application with the Flutter replica by asking the participants for differences between the two apps (further explained in Subsection ...). This methodology has the advantage of covering predetermined topics relevant to the research question while also allowing spontaneous discussion possibly leading to novel insights.
Furthermore, expert interviews are an especially useful approach for scientific explorations with no or scant preexisting theory (cf. **Bogner2009**).

The number of interviews conducted is limited as soon as no new perspectives seem to emerge from performing further interviews (Cf. ...).

### 3.3.1 Interview Preliminaries and Technicalities

The interviews are conducted with employees of apploft. They qualify as subject matter experts in the sense that they have been working in the mobile app industry for multiple years. They come from variety of professional backgrounds including UX and UI design, project management as well as software engineering. Furthermore, some interviewees have actually worked on the original app itself. This diversity among the study participants is especially relevant in order to explore a breadth of perspectives. The interviews are moderated and recorded by the author with a single interviewee per interview. For the comparison, the interviewees receive QR codes with which both apps may be downloaded. Behind each distributed code is a downloadable IPA (iOS App Store Package) binary executable file hosted by an HTTP server. These work exactly the same as any other apps downloaded from the iOS App Store. Due to the ongoing Covid-19 pandemic, the interviews are conducted through video calls and the interviewees are asked to share their iPhone screen via Quicktime player.

### 3.3.2 Interview Guideline

The interview guideline (see Appendix ...) is based on finding out perceptual differences between the iOS baseline and Flutter replica app.

Just like the performance comparison, use cases associated with particular UI features form

the basis for the evaluation:

- ...
- ...

The interviewees are asked to perform a particular use case for app A and app B. Then they are asked to detail perceptual differences between the two apps. Asking this open-ended question aims at receiving as much information possible about perceptual differences (Helfferich 2011, 182-185). Subsequently, the participant is asked to determine which of the two apps felt more natural (i.e. had a better UX). A determined tertiary response of: "A", "B" or "same" are expected. The goal of this question is to get overall impression of the usability. Both questions are asked after each use case execution of the participant. To maintain a high participant engagement during interviews, use cases are described in a more captivating way, e.g. "Please find the blue Mercedes SUV in Kickdown A [Wait until participant has found it.]. Now, please look for the black BMW convertible in the other app.". After each use case, the ordering of app A and B is swapped. E.g. if the first case starts with A, the second starts with B. This choice made as to avoid recency bias (cite ...). Furthermore, the ordering is also swapped each interview. In this way, participant X starts with app A while participant Y starts with B. Finally, after the last use case, the participant is asked to answer the two questions with regard to the entire application.

### 3.3.3 Interview Evaluation

The videos from the interviews are transcribed into a textual format and further processed using **interview coding** (cite ..). Thereby, each interview is categorized into semantic themes. These themes among all interviews are then merged into an overall theme structure - also known as code structure. This code structure is further for the evaluation of the usability hypothesis $H_U$.

# Chapter 4

# Application Design and Implementation

- what is the exact Flutter version that I've been using?

## 4.1 Kickdown Feature Presentation

Here I will present what the original features of the Kickdown app are

-> think of use specific instances where it is interesting to detail the implementation of the Flutter app - image gallery s

## 4.2 Flutter Implementation

- Thesis is not an [Ingenieurswissenschaftliche Arbeit] and thus application design and implementation will be explained from a perspective a necessary requirement to execute both the performance and UX comparison. - goal from development perspective was to get implement Flutter as close as possible to the original app in terms of UI - used as many out of the box cupertino widgets as possible for implementation - only deviated from above guideline when absolutely necessary - this was the case when implementing the following features - image gallery (zooming ability), image caching, custom modal iOS 13 presentation)s - Naturally, software implementation complexity may impede performance if unscalable algorithms or memory-inefficient datastructures are chosen. In order to prevent this bias to be introduced into the experiment, the Flutter app was implemented as closely as possible to the original iOS app. Todo: explain in what way.

- should I explain the general app architecture which was used?(MVVM + Services)

# Chapter 5

# Results

## 5.1 Performance Comparison

## 5.2 Usability Comparison

# Chapter 6

# Summary

- What are my key findings? What is their significance and implications

### 6.0.1 apploft

- Byproduct of analysis was the development of a decision guide when choosing between the development of a native and Flutter mobile app

- maybe point to future research into Flutter - Reference back to goal in introduction - What are the limitations of my research - Future Research questions - Outlook