Developing a cross-platform MVP app with React Native

A niche smartphone app to help aspiring hunters prepare for the Finnish hunting exam

Thomas Nylund

Degree Thesis
Information Technology
2020

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	24605
Författare:	Thomas Nylund
Arbetets namn:	
Handledare (Arcada):	Pekka Buttler
Uppdragsgivare:	

Sammandrag:
Detta examensarbete beskriver utvecklingen av en minimum-viable-product (MVP) applikation (app) med React Native ramverket för Android och iOS plattformarna. Dessa plattformar dominerar mobilappekonomin och för att nå så många användare som möjligt måste utvecklare skapa en app för varje plattform, vilket kräver att de lär sig två olika utecklingsekosystem, med olika programmeringsspråk och verktyg. Detta höjer utvecklingstiden och kostnaderna och gör apputveckling orimligt för många individer och småföretag. Men, cross-platform-ramverk erbjuder en lösning då de möjliggör utvecklingen av en app som kan distribueras till flera plattformar. Under utvecklingsprocessen stöder sig utvecklare ofta på mjukvaruutvecklingsteori, men hur bra stöder egentligen denna teori små mjukvaruprojekt med bara en utvecklare. Genom att utveckla en app försöker vi ta reda på hur bra mjukvaruutvecklingsteorin tillämpar sig till utvecklingen av en MVP app av en ensam utvecklare, hur användbart React Native ramverket är för en ensam utvecklar som bygger sin första cross-platform app, och hur mycket av appkoden som delas mellan de två plattformarna. Teoretiska delen diskuterar mjukvaruutvecklingsprocessteori, med en närmare titt på två utvecklingsmodeller: plan-driven- och inkrementell utveckling. Dessutom, diskuteras MVP sättet och hur det påverkar produktutveckling. Vi diskuterar också mobilapputveckling och tittar närmare på cross-platform-apputveckling och mer specifikt, React Native, för att ge läsaren insikt i teknologier inom området och varför ett ramverk valdes över ett annat. I praktiska delen diskuteras MVP appens utveckling och dess viktigaste delar, samt olika beslut som gjorts under utvecklingsprocessen. Arbetets resultat visar att mjukvaruutvecklingsteori har begränsat värde för små mjukvaruprojekt i stil med projektappen och att mjukvarukokböcker är mer nyttiga för oerfarna utvecklare. Dessutom, visade sig React Native vara ett mycket bra ramverk för en ensam utvecklare med tidigare erfarenhet av JavaScript och React. Ramverket låter en utvecklare utnyttja mycket av sin existerande kunskap för att kostnadseffektivt utveckla en MVP app som kan lanseras nativt på två plattformar, vilket gör det möjligt för lönsam utveckling av även nischappar. Till slut så delade projektappen nästan all kod mellan plattformarna.

Nyckelord:	Cross-platform, React Native, Expo, minimum viable product (MVP), mobilapplikationer, mjukvaruutvecklingsprocessen, mjukvaruutvecklingsmodeller
Sidantal:	
Språk:	Engelska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	24605
Author:	Thomas Nylund
Title:	
Supervisor (Arcada):	Pekka Buttler
Commissioned by:	

Abstract:
This thesis describes the development of a minimum-viable-product (MVP) app with the React Native cross-platform framework for the Android and iOS platforms. These platforms dominate the app economy and to reach as many users as possible, developers have to create one app for each platform. This requires them to learn two very different development ecosystems, with different programming languages and tools. This increases development time and cost, making it unfeasible for many individuals and small businesses. Cross-platform frameworks offer a solution by allowing developers to create one app that can be distributed on multiple platforms. When developing apps, developers often rely on software development theory to support them during the development process, but how well does the theory support smaller projects that might only have one developer. By developing an app, the objective of this thesis is to find out how well software development theory supports a sole developer building an MVP app, how useful the React Native framework is for a sole developer building their first cross-platform app, and how much of the code can be shared across the two target platforms. The theoretical part of the thesis will look at software development process theory, with a closer look at two software development models: plan-driven and incremental development. In addition, we will discuss the MVP approach and how that influences product development. We also discuss mobile app development, with a closer look at cross-platform apps and React Native to give the reader an understanding of the available technologies in the space and why one framework was chosen over another. In the practical part, we will walk the reader through the development of the app and its key elements, and many of the decisions that were made in the process. The results of the work show that software development theory has limited value for small-scale software projects, like the case app, and that software cookbooks are more helpful to inexperienced cross-platform app developers. In addition, we found that React Native is a great framework for a sole developer familiar with JavaScript and React. It allows the developer to leverage much of their existing knowledge to develop a low-cost MVP smartphone app that can be published natively to two platforms, allowing the development of even niche applications to be profitable. In the end, the case app shared almost all of the codebase between the two target platforms.

Keywords:	Cross-platform, React Native, Expo, minimum viable product (MVP), mobile applications, software development process, software development methodologies
Number of pages:	
Language:	English
Date of acceptance:	

CONTENTS

List of abbreviations 7

1 Introduction 8

1.1 Background 9
1.2 Objective / Purpose 10
1.3 Structure of the thesis 11
1.4 Limitations and delimitations 12

2 Software Development Process 12

2.1 Software specification (requirements engineering) 13
2.2 Software design and implementation 15
2.3 Software validation 18
2.4 Software Process Models 19
2.4.1 Plan-driven development 20
2.4.2 Incremental development 21
2.4.3 Minimum viable product (MVP) 24

3 Mobile Application Development 25

3.1 Native application development 27
3.2 Cross-platform application development 28
3.3 Cross-platform vs. native development 29
3.4 Cross-platform technologies 30
3.4.1 Hybrid cross-platform apps 30
3.4.2 Cross-platform native apps 31
3.5 Why I chose React Native for developing the app 33
3.5.1 A closer look at React Native 33

4 Building The MVP With React Native 34

4.1 Preparation 35
4.1.1 Requirements 35
4.1.2 Tools 35
4.1.3 Installation and setup 36
4.1.4 The beginning 37
4.2 Development 38
4.2.1 Buttons 38
4.2.2 Styling the components 39
4.2.3 Screens and navigation 39
4.2.4 Quiz logic and questions 41
4.2.5 ResultsList 43
4.2.6 State 43

4.2.7 User experience-enhancing functionality ... 44
4.3 Testing ... 46
4.4 Code sharing between platforms ... 47
4.5 Reflecting on my development process ... 47
5 Conclusion ... 49
References ... 53
Appendix ... 56
6 Swedish Summary ... 56

Figures

Figure 1. The requirements engineering process (Sommerville 2016, p. 55). 14
Figure 2. A general model of the design process (Sommerville 2016, p. 56). 16
Figure 3. The waterfall model (Sommerville 2016, p. 47). 20
Figure 4. Incremental development (Sommerville 2016, p. 50). 22
Figure 5. How hybrid cross-platform frameworks work (Duffy 2018). 30
Figure 6. Hybrid app architecture (Duffy 2018). 31
Figure 7. How cross-platform native frameworks work (Duffy 2018). 32
Figure 8. Two types of cross-platform native apps (Duffy 2018). 32
Figure 9. Screenshot of app buttons. 38
Figure 10. Example of React Native core component importing. 38
Figure 11. Screenshot of button styling. 39
Figure 12. A few screenshots of screens from the app. From left to right, Menu, ImageQuestions, and ResultsView. 39
Figure 13. Sequence diagram of the app navigation flow. 40
Figure 14. Image of NavigationBar component. 41
Figure 15. ResultsView component and its props. 41
Figure 16. Data structure of text-based question. 41
Figure 17. Data structure of image-based question. 42
Figure 18. Network connection notification. 44
Figure 19. Image of ActivityIndicator. 45
Figure 20. Image of app icon on iOS 46
Figure 21. Image of iPhone SE and Nexus S emulators 47

Tables

Table 1. The principles of agile methods (Sommerville 2016, p. 76). 24
Table 2. The difference between native and cross-platform app development (Manchanda 2019). 28

LIST OF ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command-line Interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
MVP	Minimum Viable Product
NPM	Node Package Manager
OS	Operating System
OTA	Over-The-Air
SDK	Software Development Kit
UI	User Interface
UX	User Experience

1 INTRODUCTION

If there is a simple everyday problem that could be solved by technology, then consumers can probably find a smartphone app that tries to offer a solution. This is why Apple filed to trademark the slogan “There’s an app for that” (Chen 2010) in 2009. Ten years later (2019) the app economy has exploded with consumers downloading a record 204 billion apps during the year, with worldwide App Store spending reaching $120 billion, up 2.1x from 2016.

Over the past decade, consumers have gotten more and more used to looking for an app no matter how niche the desired service might be, further fueling demand. One such niche could be an app that helps aspiring hunters prepare for the Finnish hunting exam, the passing of which is required to legally hunt wild game in Finland. However, for such a niche app to make business sense, the development would need to be low-cost and simple.

While the iOS and Android app stores allow apps to be distributed to billions of consumers at very low additional cost after creation, product development bears a lot of risk because of the time and resources it requires, further exacerbated by the unpredictable nature of how consumers might receive a product. To reduce such risk, simple working prototypes are often developed and released to market quickly. This allows users to test the product early and provide real feedback, which then guides further development in small increments, thereby ensuring that the final product meets the users’ requirements.

However, such an approach creates a need for both development tools that facilitate fast delivery of working software, and a software development process able to support it. Cross-platform development frameworks promise to allow the development of an app that can run on several platforms, cutting down on needed technological know-how and development time, thereby also promising to make app development more accessible to smaller teams.

But would such a cross-platform development framework allow a single relatively inexperienced programmer to produce a cross-platform mobile app and publish it on both the iOS and Android app stores. Could such a development project be sufficiently low-cost for targeting even small niches? Can there really be an app for even that?

1.1 Background

In the summer of 2007, Apple launched the iPhone with the iOS operating system (OS) (Clark 2012). One year later, Google launched its own smartphone OS called Android. Shortly after, in July 2008, Apple launched the App Store with 552 apps (Strain 2015). Many consider this having given birth to the so-called app economy (Strain 2015, Wikipedia 2020a). In the following years, several smartphone manufacturers, like Google, RIM, and Microsoft, launched their own app stores to make their devices more attractive to consumers (Clark 2012). Today, Apple iOS and Google Android dominate the market for smartphones and apps with 99% of devices using one of their OSs (O'Dea 2020). Hence, covering these two platforms is essentially sufficient to reach the whole market.

Today, no-one doubts the crucial role apps play in allowing users to make maximum use of their smartphones. On average, people use their smartphones for three hours and forty minutes per day (App Annie 2020). That time is split among the following categories: social media and communications 50%, video and entertainment 21%, gaming 9%, and other 19%. Gaming also accounted for 72% of app store spending, which indicates that smartphone users are quite willing to pay for something which amuses them or allows them to pass the time (App Annie 2020). Increasing usage, spending, and app downloads tell us that users prefer to use services that are made specifically for handheld devices.

The app economy's large potential market, low barriers to entry, almost free distribution, and frictionless consumption are the main arguments for why an app idea with a sound business case should be strongly considered. Moreover, a 'sound business case' need not be big business – even a relatively niche product can be lucrative if costs can be kept low. All these factors contribute to creating opportunities for small businesses which, despite their scarce resources, can employ technologies facilitating fast and efficient delivery of software to multiple platforms. Together with production philosophies like Agile and Lean, that embrace incrementally building out product features in order of their necessity, it is possible to create cost-efficient products that offer clear utility to their users – no matter whether the potential user base is numbered in millions or mere thousands.

Annually, over 6000 people pass the Finnish hunting exam (Riistainfo s.a. a). Taking the exam costs 20€, and it consists of 60 questions, out of which 15 are image-based animal

identification questions and 45 are text-based multiple-choice questions (Riistainfo s.a. b). Passing the exam allows for no more than 8 wrong answers. In 2017 and 2018 the success rate of the exam was 55% and 69%, which indicates that many people have to try multiple times in order to pass (Mikkola 2018).

For years I have studied languages on my smartphone because it is so convenient. There is no need to carry around any books and I can easily study with the help of a device I almost always have with me. This inspired me to look for an app to help me study for the hunting exam, but I was disappointed when I did not find one. There is a variety of learning material available, both digital and print, but no smartphone app. This simple but unmet need coupled with the requirement of a final project gave me the idea to develop an app and use it as a case for my thesis.

1.2 Objective / Purpose

The objective of this thesis is to utilize cross-platform technologies together with agile product development philosophies to test their suitability for a business case that requires app development to be low-cost, quick, and simple. To test this, we are going to build a minimum viable product (MVP) app that is intended to be released in the iOS and Android app stores. "Unlike a prototype or a concept test, an MVP is designed not just to answer product design or technical questions. Its goal is to test fundamental business hypotheses" (Ries 2011, pp. 93-94). The app would initially be a free quiz style app that mimics the structure of the hunting exam. Based on the feedback gathered on the app it would then be further developed. If it proves valuable to its users, it could then be monetized.

The most practical way to target both the iOS and Android platforms is to use a cross-platform framework. Cross-platform frameworks allow the development of one app that can run on several platforms, cutting down on needed technological know-how and development time. This will be discussed in more detail in section three.

The goal is to develop an MVP app for the earlier described business case. By developing this app, we will investigate how well the React Native framework is suited for this case and if it fulfills its promise of "learn once, write anywhere", allowing for lower development time and needed know-how to build native smartphone apps (React Native s. a. a).

We will investigate how much of the case apps codebase is shared across the platforms. And, we will also examine how well the existing software development theory supports the development of a simple MVP, like the case app, that is built by a sole developer.

The questions answered in this thesis are:
Can a sole developer utilize a cross-platform framework to develop a low-cost MVP app for both the iOS and Android platforms?
How much of the codebase in the MVP is shared across the iOS and Android platforms?
How well does existing software development process theory support the development of an MVP by a sole developer?

1.3 Structure of the thesis

This thesis is divided into a theoretical and a practical part. The theoretical part of the thesis is divided into two sections. Section two, which is the first of the theoretical sections, will look at the three main stages of the software development process: requirements specification, software design and implementation, and software validation. After that, we will touch on two software process models, plan-driven and incremental. Finally, we will look at the how, why, and when of the MVP approach. This section forms the underlying theoretical foundation for the development of the app.

Section three introduces mobile application development, explains the differences between native and cross-platform mobile development, and then focuses on different types of cross-platform apps, with a closer look at the React Native framework. This section covers the technologies that have influenced and been used in the development of the MVP.

The practical part of the thesis focuses on the development of the MVP app using the React Native framework. This part will not be a step-by-step guide of the development process. Instead, it will showcase the app and give an overview of the development process by discussing the tools and libraries that were used in the implementation, and the technological decisions that were made to arrive at the final product.

1.4 Limitations and delimitations

The thesis focuses on the three first stages of the software development process, the development of a cross-platform MVP app, and the technologies involved. As mentioned earlier, the app will be developed with React Native and other smartphone development technologies will only be discussed briefly. This does not mean that React Native is the best framework for developing the case application, but rather that it was the best option in light of the specific skills, background, and preferences of the author.

The thesis covers the development process of the case app to the stage where the app could be released to the app stores, but it will not cover how an app is released to app stores, the subsequent feedback that is gathered, or the evolution of the app after the launch. The evolution of an app can take much longer than developing the initial version and it is therefore out of the scope of this thesis.

* * *

Aside from building a simple web page with a personal blog on it, building software products can quickly become a very complex task that takes several weeks or months, and hundreds of man-hours to complete. To better understand how software products are built, we should start by looking at the software development process and the common stages developers go through when building software products. This is what we will cover next.

2 SOFTWARE DEVELOPMENT PROCESS

The process used when developing software from concept to product is called the software development process (Sommerville 2016, pp. 22-23). This process involves a systematic execution of activities that help developers create high-quality software. Sommerville (2016, p. 23) lists four activities involved in all software development processes:

1. *Software specification*, where customers and/or end-users together with engineers define the software that is to be developed. They specify what the software should and should not do.
2. *Software development*, where the software is designed and implemented.

3. *Software validation*, where the software is inspected to ensure that it meets all the requirements of what has been ordered.
4. *Software evolution*, where the software is maintained and improved to meet the changing needs of the customer and the market.

The above-mentioned activities can be very complex and greatly vary in workload depending on the project. They often also involve many sub-activities such as project planning, architecture design, and unit testing. (Sommerville 2016, p. 44)

In practice, most software companies tailor the development process to fit their organizational characteristics, like the skills and knowledge of their software developers, and the requirements of the projects that they usually develop. But they largely still rely on the same process, executing the same fundamental activities in the same order. Next, we are going to look at these activities in more detail. (Sommerville 2016, p. 45)

2.1 Software specification (requirements engineering)

Software specification is the process of gathering information about the end-users' and/or customers' wants and needs through discussions, investigation of tasks, workflows, processes, and existing technological solutions (Sommerville 2016, p. 54, Stephens 2015). The information is then analyzed with the objective of figuring out what services are required from the software and identifying restrictions for its development and operation (Sommerville 2016, p. 54, Stephens 2015). Both Sommerville (2016, p. 54) and Stephens (2015) emphasize that this is a crucial part of the software development process as a whole because any mistakes at this stage will affect the design and implementation of the software.

For a requirement to be useful, it should be expressed in a statement that is clear and easy to understand, with an explicit and verifiable request so that everyone involved easily comprehends what it is about. Stephens (2015, chapter 4) explains "They[requirements] can't be open-ended statements such as, "Process more work orders per hour than are currently being processed." How many work orders is "more?"[…]". This can be achieved by avoiding unnecessary use of technical terms, abbreviations, management speak, and

open-ended statements, so that there is little room for misinterpretation on the individual level.

The specification process (see image below) aims to produce a requirements document that specifies a software system that meets the needs of the customers and end-users. The requirements are usually documented on two levels: the end-users and customers want to know the high-level requirements, while the developers need more detailed documentation. (Sommerville 2016, p. 54)

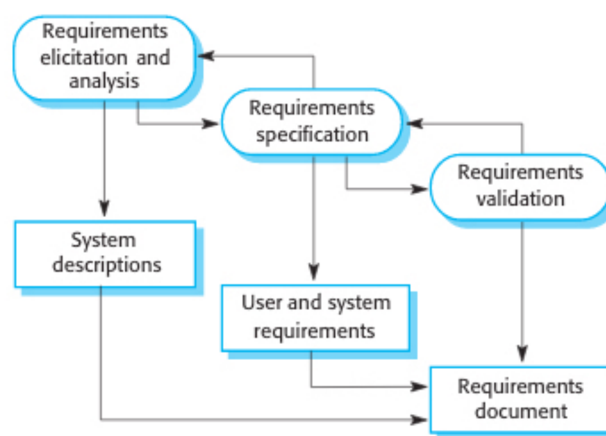

Figure 1. The requirements engineering process (Sommerville 2016, p. 55).

According to Sommerville (2016, p. 55) the software specification process includes the following three main activities:

1. *Requirements elicitation and analysis*. This entails the collection of software requirements by researching existing systems, discussions with customers and suppliers, workflow analysis, and so on. System models and prototypes are often developed at this stage to get a better understanding of what the users really need.
2. *Requirements specification*. This requires that the information that was collected and analyzed in the previous activity is transferred to a document that defines a set of requirements. The document often contains two types of requirements. The first type is user requirements, which are descriptions of what the end-users and customers want and need from the software. The second type is system requirements that describe, in detail, the features that the system should have.

3. *Requirements validation*. This encompasses that the requirements are checked to confirm that they are realistic, consistent, and complete. This activity often reveals errors that then have to be corrected.

New requirements can come to light at any stage of the requirements specification process (Sommerville 2016, p. 55). Figure 1 provides an overview of the entire requirements specification process and clearly shows how one moves back and forth between the different activities. The process results in a requirements document that supports the design and development of the software system (Stephens 2015). The format and level of detail of the specified requirements depend on the size and complexity of the project. Stephens (2015, chapter 12) explains:

> For a large traditional project, the specification might include hundreds of pages of text, charts, diagrams, and use cases. For a small project that you are writing for your own use, the specification might be all in your head, and you might "write" it while walking the dog or singing in the shower.

In agile development methods, requirements specification is not considered a separate activity from development. At the beginning of each cycle of the development process, requirements are informally specified before developers start programming. Requirements are collected from end-users who also prioritize them. The development team actively communicates with the end-users throughout the development process to make sure the requirements stay up to date and that they are interpreted correctly. (Sommerville 2016, p. 55)

2.2 Software design and implementation

Software development can be seen as a process where an idea or concept is broken down into ever smaller parts until the parts are so small that they can easily be programmed (Stephens 2015). The goal of the design and implementation stage is to develop a working piece of software that can be delivered to the customer (Sommerville 2016, p. 56). Plan-driven development methods treat design and development as separate activities often conducted by different people, whereas Agile development methods see them as being interleaved. Sommerville defines software design as (Sommerville 2016, p. 56):

> A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used.

Software programming often starts before the design is finished. A group of developers and designers develop the design in a step-by-step process. Each step brings more detail to the design, and it is not uncommon to have to repeat design activities since they are interconnected and interdependent (see figure 2). As the design work progresses, new information is generated, which might affect earlier design decisions and details. This leads to a lot of rework during the design process while working toward completeness. Figure 2 provides an overview of what a design process might look like (Sommerville 2016, p. 56)

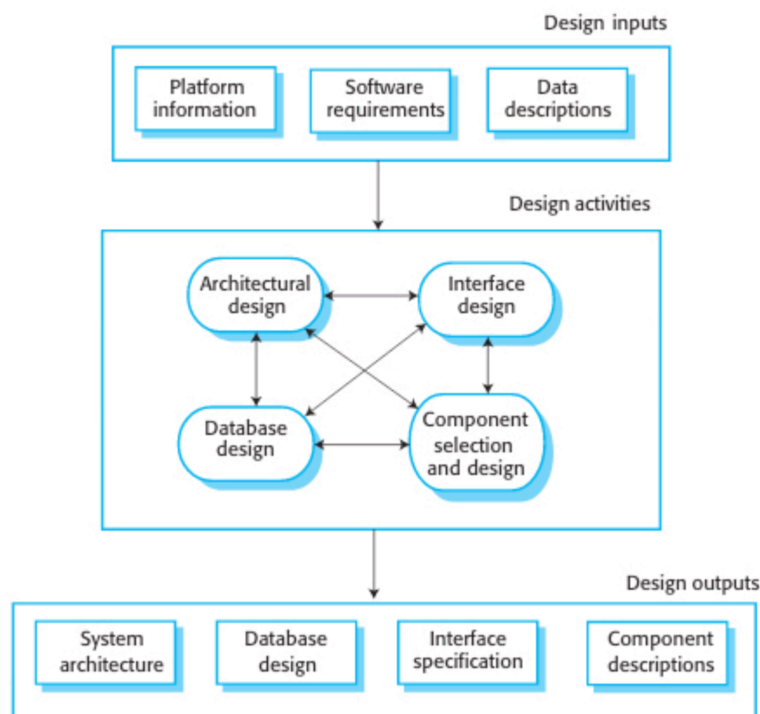

Figure 2. A general model of the design process (Sommerville 2016, p. 56).

Software systems rarely operate in a silo, but instead, they usually interface with other software systems, e.g., OSs, databases, middleware, and software applications (Sommerville 2016, p. 57). Together, they make up what Sommerville calls the software platform, which is the environment where the software system will be run. A good understanding of the software platform is vital to the design process because developers have to decide how the system should integrate with its environment.

Sommerville (2016, p. 57) highlights that design process activities vary depending on the type of software system, but he describes four common activities as follows:

1. *Architecture design*. The system structure is visualized as a whole: what are the main components that make up the system, their relationship to each other, and how are they distributed.
2. *Database design*. During this process, data structures of the system and how they should be defined in the databases are designed.
3. *Interface design*. The interfaces of the different system components are defined. Sommerville emphasizes that the interface specification has to be unambiguous. A component with a well thought out interface can be used by other components without them knowing how the component will be implemented. Components with well-defined interface specifications can be designed and developed separately.
4. *Component selection and design*. In order to save time, one might look for reusable components. However, if they cannot be found, then new components have to be developed. At this stage the design might be a simple component description, which means that the implementation details are left for the developers. There might also be a list of modifications that have to be made to a reusable component.

The above-mentioned activities result in formal or informal design documents or sometimes directly in written code, which might be the case with Agile development methods. Agile methods seldom use formal design documents. Instead, the design might be written on whiteboards, or in developers' notebooks or laptops. These formal/informal documents work as the foundation for software programming. (Sommerville 2016, 56-57)

Programming is a separate activity, and Sommerville (2016, p. 58) accentuates that there is not a set process that developers would usually follow. Some start with the components they are familiar with, while others start with the unfamiliar components because they already have an idea of how the familiar components work.

The next activity in the development process, which is testing, starts before programming has finished (Stephens 2015). When a developer has finished programming a component, they test the code to check it for errors (Sommerville 2016, p. 58). Sommerville (2016, p. 58) highlights the following difference between testing and debugging:

> Finding and fixing program defects is called debugging. Defect testing and debugging are different processes. Testing establishes the existence of defects. Debugging is concerned with locating and correcting these defects.

Stephens (2015) emphasizes that it is best to test the code early and often, because it is easier to find and fix problems right after the code has been written.

2.3 Software validation

The objective of software validation is to demonstrate that the software system conforms to the specifications, meets the requirements of the customer, and works under the specified circumstances. (Sommerville 2016, p. 58, Stephens 2015)

Typically, software systems are validated through tests with test data. Alternatively, one might schedule controls that include inspection or review of the system or code at different stages throughout the development process. (Sommerville 2016, p. 58)

Small software systems can easily be tested as a whole. However, larger systems have to be tested in phases (Sommerville 2016, p. 58). Sommerville describes three testing phases in more detail (Sommerville 2016, p. 59):

1. *Component testing*. The developers test the components that make up the system. Each component is tested independently. A component might be a function, a class, or some sort of system feature that uses several functions and/or classes.
2. *System testing*. The components are put together to create a complete system. The system is then tested as a whole to check if the different components have been integrated correctly. The objective is to demonstrate that the system works and meets the requirements specification.
3. *Customer testing*. At this phase, the customer tests the software before finally approving it for real use. The customer performs real work tasks with the software and flags any inadequacies that weren't discovered during tests with test data. This phase might reveal faults in the requirements specification, which could result in functionality that does not meet the customers' needs.

In the first phase, the components of the system are tested separately, after which the system is tested as a whole, and finally, the customer tests the system with real work tasks and customer data. The aim is to discover errors and inadequacies as early as possible,

which is why most software development methods plan testing phases into the development process beforehand. Component testing is usually part of the daily work of a developer as they test code snippets while programming a component. This is a natural solution since developers know how a component should work when they have programmed it. (Sommerville 2016, p. 59)

When an error is discovered it has to be fixed. But this can have an unforeseen negative impact on other system components, which means that one might have to redo tests that were completed earlier in the process. This leads to a naturally iterative process where information is discovered that then has to be considered in the context of earlier stages of the process. (Sommerville 2016, p. 59)

With Agile development methods that develop software in versions, it is not uncommon that each version of the software would go through all of the three above-mentioned testing phases. This lowers the risk of the final software product not satisfying the customer's needs. (Sommerville 2016. p. 59)

2.4 Software Process Models

Software process models, more commonly referred to as software development methodologies, represent the activities included in the process in a simplified abstract manner. These models aim to clarify in what order the different development activities are carried out, but they do not specify who executes them. (Sommerville 2016, p. 45)

A variety of different process models exist, and they all have their specific strengths and weaknesses depending on the circumstances. Therefore, it is important for a developer to be able to pick the right model for the prevailing situation (Sommerville 2016, p. 22). "The right process depends on the customer and the regulatory requirements, the environment where the software will be used, and the type of software being developed" (Sommerville 2016, p. 46).

Stephens (2015) emphasizes that the models are similar in many ways, but the differences arise in how they handle the development activities. Based on these differences, the models can be divided into categories. Stephens (2015) splits them into predictive and iterative

models. Poppendieck & Poppendieck (2006) define them as being deterministic or empirical. Sommerville (2016, p. 46) describes two types of general process models that fit custom software projects: the waterfall model and incremental development. We have chosen to categorize them as plan-driven and incremental development approaches. Next, we will look at the general idea behind these two approaches, how they differ, what their advantages and disadvantages are, and then we will discuss how they might work for a single person or small development team with scarce resources.

2.4.1 Plan-driven development

In practice, a plan-driven process entails that one would prepare and schedule all the development activities before the actual programming of the software is started (Sommerville 2016, pp. 47-48). During the development process, each activity is seen as its own phase (see figure 3). The completion of a phase results in a document, code, or graphics that then have to be approved by the customer before moving on to the next phase. This type of approach works well for hardware with high manufacturing costs. However, with software the development activities often mingle and exchange information, and Sommerville emphasizes that the process is seldom linear.

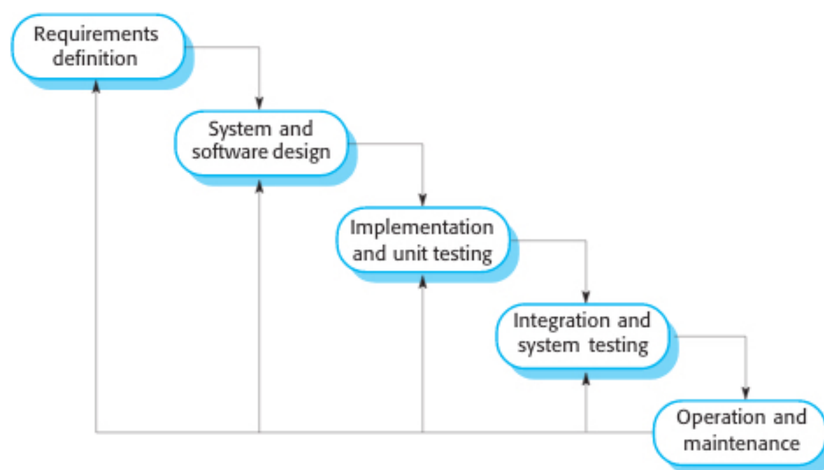

Figure 3. The waterfall model (Sommerville 2016, p. 47).

Problems with the plan-driven process arise from its requirement to plan and decide things beforehand, which works under the assumption that software developers have all the necessary information early in the development process (Stephens 2015). Stephens points

out that it is often hard to know exactly how the software should work, what functionality it needs, and how those features should be built. Combined with new technology, a plan-driven approach is even more difficult to apply. However, a plan-driven approach can work really well in some situations. Stephens (2015, chapter 12) explains:

> For example, predictive models work well when the project is relatively small; you know exactly what you need to do, and the timescale is short enough that requirements won't change during development.

While a development activity is underway, it continuously generates new information. This information might have unintended consequences for earlier activities, which then lead to modifications in earlier plans and possibly rework. If the modifications are not approved by stakeholders, programmers are forced to circumvent or avoid an issue altogether, which leads to a sub-optimal solution. (Sommerville 2016, p. 48)

Sommerville (2016, pp. 48-49) highlights that rework is always required at the final stages of the development process when errors, inadequacies, and needs for additional features arise after the software is tested or in use. Also, to ensure that the software remains useful, it will have to be maintained and expanded. This, in turn, means that several of the development activities have to be repeated, which is why the plan-driven process is not suited for situations where the requirements change quickly and there are several unknown variables.

2.4.2 Incremental development

Incremental development builds on the idea of an early implementation or prototype that is used to gather feedback from stakeholders (Sommerville 2016, pp. 49-50). Stephens (2015, chapter 4) describes a prototype as follows:

> A prototype is a mockup of some or all of the application. The idea is to give the customers a more intuitive hands-on feel for what the finished application will look like and how it will behave than you can get from text descriptions such as user stories and use cases.

The feedback then guides the development of additional versions of the software until the system meets all requirements. Instead of executing development activities separately, they actually coincide with rapid information exchange between them (see figure 4). (Sommerville 2016, pp. 49-50).

The concept of using a prototype to gather feedback is closely related to the approach of using an MVP to test a business hypothesis. MVPs are covered in chapter 2.4.3 later on in this section.

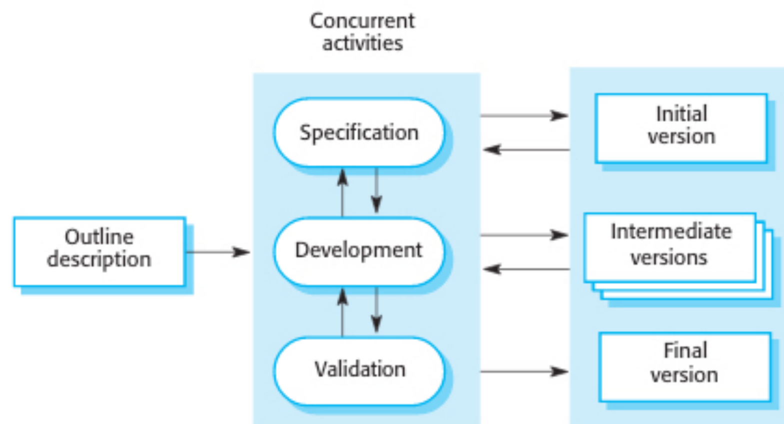

Figure 4. Incremental development (Sommerville 2016, p. 50).

Typically, the first versions of the software contain the functionality that is most valuable to the customer (Sommerville 2016, p. 50). Each round of development goes through requirements specification, development, and validation that then lead to new functionality and a new version. Stephens (2015) points out that functionality is often developed in small steps, which keeps the increments short and allows the stakeholders to see and possibly test the software multiple times throughout the development process. This leads to frequent feedback and modifications that facilitate iterative improvements, which increases the likelihood of the software creating value for the customer (Sommerville 2016, p. 50).

Sommerville (2016, p. 50) highlights three benefits of incremental development compared to the waterfall model:

1. Compared to the waterfall model, incremental development encourages continuous refinement of the requirements specification. This reduces the amount of rework and lowers the implementation costs.
2. Having a working prototype or implementation of the software makes it easier to collect feedback from stakeholders. A lot of people might find it hard to give feedback based on written documents or static graphics.
3. As soon as a version of the software is considered to be valuable enough, it can be delivered to the customer for use. This allows the customer to gain value from the software much earlier than would be possible with the waterfall model.

Incremental development does not imply that a complete enough version of the software has to be delivered to the customer so they can start using it right away. The functionality in a new version of the software is often demonstrated with the sole purpose of gathering feedback from the end-users, which then allows development to continue.

Incremental development is not without its risks. Sommerville (2016, p. 51) brings forth two problems from a leadership perspective:

1. The process is often informally documented and difficult to follow. Management requires consistent deliverables to measure progress. However, sometimes it is not worth putting down too much time on documentation for each version if it does not support the development work.
2. The quality of the system architecture has a habit of degrading as the program versions increase. Simultaneously, continuous changes and additions of new functionality can lead to messy code. This makes the implementation of subsequent versions more difficult and expensive. To counteract this, it is recommended to regularly refactor the code, which means engineers would dedicate time specifically to improve code structure and readability.

Incremental development is a fundamental part of Agile development methods (Sommerville 2016, p. 50). Agile methods rely on a shared set of principles (see table 1) to guide the development process in the fast-paced, ever-changing business environment (Sommerville 2016, pp. 75-76). They are best suited for situations with high uncertainty, where software requirements often change during the development process. This is common in business application and software product development, which is why incremental development has become so popular (Sommerville 2016, p. 50). Agile works best when developing new software in small co-located teams where the focus is more on design and implementation and less on documentation (Sommerville 2016, p. 89). However, many modern software companies have development teams that are distributed across the globe. In addition, Agile methods are harder to adapt to "the legal approach to contract definition that is commonly used in large companies" (Sommerville 2016, p. 89).

Table 1. The principles of agile methods (Sommerville 2016, p. 76).

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
People, not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

Small start-up companies can work informally with short-term plans, but larger companies must have long-term plans and budgets for investing, personnel, and business development. This pushes software development in larger companies to also think more long-term. To address these challenges, many large Agile software development projects have combined elements of the plan-driven and the Agile approach. If they end up being more plan-driven, or more Agile, depends on the characteristics of the organization. (Sommerville 2016, p. 91)

2.4.3 Minimum viable product (MVP)

An MVP is the simplest version of a product that is deemed good enough to be released to customers so they can provide feedback on the product through their usage (Ries 2011, pp. 93-94). "Unlike a prototype or a concept test, an MVP is designed not just to answer product design or technical questions. Its goal is to test fundamental business hypotheses" (Ries 2011, pp. 93-94). If the product proves valuable to its users, their feedback is then used to improve existing features and add new ones (Baker 2018).

In general, product development is seen as a risky process because of the costs accrued during development and the unpredictable success after a product launch. An MVP tries to reduce this risk with an early release to quickly test customers' reactions and gather feedback. By using this approach, businesses can avoid a scenario where a product is developed to perfection at a high cost, but then ends up getting rejected by the customers when released to the market. (Baker 2018)

For a few reasons, the MVP approach suits small businesses and entrepreneurs especially well. First, an MVP does not require big resources, brand recognition, or reach into multiple markets, all of which small businesses and entrepreneurs rarely have. Second, a competitive advantage of small businesses is that they can often adapt to feedback and changes quicker. Third, a good MVP might generate revenues at an early stage, which can be crucial for a cash strapped small business or entrepreneur to be able to continue improving the product. (Baker 2018)

* * *

There are common activities that individuals and teams execute in the process of developing software products. And, the approach used for executing these activities depends on the circumstances of the situation at hand, including the resources of the organization, the characteristics of the product, and more. But, are there also common technologies that developers use to build their software products? The short answer is no. There are many different tools and languages available that a developer might consider, all of which come with their own circumstantial strengths and weaknesses. In the next section, we will take a high-level look at some of the technologies that relate to the context of an MVP smartphone app.

3 MOBILE APPLICATION DEVELOPMENT

Mobile app development refers to the process of developing an application that targets mobile devices, this includes smartphones and tablets but not laptops. From an app development perspective, a device with a small screen that is almost always with the user and is operated mostly with fingers comes with its own set of challenges. These mainly relate to user experience, user interface, user input technology, and differences in platform technologies (Mushtaq et al. 2016, Hansson & Vidhall 2016, p. iii).

1. The *user experience* on a mobile device is much different than on a laptop or desktop. While computers and laptops are used for longer periods of time to do complex work, mobile devices are often used in short spurts to kill time, look up information, communicate, and so on. When the user interacts with the device, they want to do something specific, very quickly, without having to jump through

too many hoops. The way that the user interacts with the device has a strong influence on how apps are designed. (Mushtaq et al. 2016)

2. *User interface.* Navigating and reading content is much harder on smaller screens, so it is common that menus have to be redesigned and content thinned out and restructured to make it more readable. The mobile app user interface should be simple, intuitive, and easy to use, with the most relevant features highlighted and easily accessible. Maximizing the use of the limited screen of mobile devices gives user interface design on mobile apps greater importance and difficulty. (McWheter & Gowell 2012, Mushtaq et al. 2016)
3. *User input technology* on mobile devices is usually restricted to a few buttons and a screen. “Mobile devices have pioneered the use of non-keyboard “gestures” e.g., touch, shake and pinch as effective and popular methods of user input” (Mushtaq et al. 2016, p. 1098). High-quality apps need to support the different methods of user input to stand out and provide a great user experience. (Mushtaq et al. 2016)
4. *Platform differences.* The mobile application market is dominated by two platforms, Google's Android and Apple's iOS. A platform refers to “the hardware/software environment for laptops, tablets, smartphones, and other portable devices” (PC Mag s. a.). Each platform uses different programming languages, development tools, design guides, and app stores (Abbott & Djirdehh 2020 p. 8, Android s. a., Apple s. a.). Users are split between the two platforms so reaching the maximum number of users requires that an app is developed for both platforms. However, the platform differences make the development of two separate apps time-consuming and costly. Both Hansson & Vidhall (2016, p. iii) and Abbott & Djirdehh (2020, p. 8) highlight the differences between these platforms as a major challenge for mobile app developers.

When a mobile app is developed using one of the programming languages and dedicated tools of the platform, it is considered a native app. But to simplify mobile app development and reduce costs, technologies have been developed that make it possible to create one app that can be distributed on multiple platforms (Hansson & Vidhall 2016, p. 1). Such apps are called cross-platform apps. We will cover both native and cross-platform app development in the following chapters.

3.1 Native application development

Traditionally, apps have been developed using tools that target a specific platform, like iOS or Android (Bjørn-Hansen et al. 2019, p. 1). This is called the native development approach, and Bjørn-Hansen et al. explain it is "pointing to the use of development environments, Software Development Kits (SDKs), and programming languages native to the target mobile platform" (Bjørn-Hansen et al. 2019, p. 1). If developing for iOS, one can program in languages Swift and Objective-C, and for Android, using languages Java and Kotlin (Abbott & Djirdehh 2020, p. 8).

Different platforms use different programming languages, but the differences do not end there. Abbott & Djirdehh (2020, p. 8) emphasize that one of the biggest challenges for native app development teams is to get familiar with all the differences in technologies. They reiterate that (Abbott & Djirdehh 2020, p. 8):

> These platforms have different toolchains. And they have different interfaces for the device's core functionality. Developers have to learn each platform's procedure for things like accessing the camera or checking network connectivity.

As a result, companies usually need separate teams with specialized knowledge for each platform (Duffy 2018). This would make the development of an app for multiple platforms time-consuming and expensive - and creates the need to be able to develop apps that work on multiple platforms (Hansson & Vidhall 2016, p. 1). This is possible with cross-platform development technologies, which we will discuss in chapter 3.2.

As we can see from table 2, the high cost of development and reduced code usability are the main drawbacks of native apps. If the team has the knowledge, time, and resources to develop native apps for each platform they want to target, then the platform software-development-kit (SDK) will ensure that they can access all device APIs that they need. They will also be able to utilize the user interface components of the device to its fullest and keep the app consistent with the guidelines and latest functionality of the platform. In addition, native apps always have the upper hand when it comes to performance, even though it might not be noticeable to the human eye, as they can access the device API directly without going through an additional software layer.

Table 2. The difference between native and cross-platform app development (Manchanda 2019).

Parameters	Native Apps	Cross-Platform Apps
Cost	High Cost of development	Relatively low cost of development
Code Usability	Works for a single platform	Single code can be used for multiple platforms, for an easy portability
Device Access	Platform SDK ensures access to device's APIs without any hindrance	No assured access to all device APIs
UI Consistency	Consistent with the UI components of the device	Limited consistency with the UI components of the device
Performance	Seamless performance, given the app is developed for the device's OS	High on performance, but lags and hardware compatibility issues are not uncommon

3.2 Cross-platform application development

In general, cross-platform development aims at simplifying app development by reducing the required knowledge needed for development, the number of development tools used, and the number of codebases that have to be developed and maintained. In cross-platform development, developers can use the same programming language and toolset for developing an app that can be distributed on multiple platforms. This allows for smaller teams and makes development more affordable. The codebase is shared between platforms and that makes it easier to update and maintain as developers only need to "modify a single set of assets and those assets are propagated to each platform your app supports" (Duffy 2018). (Duffy 2018)

Cross-platform technologies are not maintained by the platform companies themselves but by third-party organizations. There are more than a dozen different technologies and they all work differently, however, the apps they produce are generally one of two types: "those that use web technologies, and those that are translated to native apps" (Duffy 2018). We will take a closer look at different types of cross-platform apps in chapter 3.4.

The shortcomings of cross-platform technologies generally relate to app performance, look and feel, and access to native features (Hansson & Vidhall 2016, p. 2). The performance can suffer because the apps are not utilizing the OS's native environment but accessing it through an additional layer (Hansson & Vidhall 2016, p. 6). Apps that rely on

the OS's core libraries, like the graphics system, might be better to develop natively to ensure access to all graphical components the OS has to offer (Duffy 2018). If the app needs to access the device's low-level hardware such as the camera, GPS sensors, or accelerometer, there are often inadequacies in cross-platform technologies, and unrestricted access is not guaranteed.

Cross-platform technologies offer a lot of flexibility when it comes to development tools, but they often have dedicated tools for building the app when the app has been completed and is ready to be launched (Duffy 2018)." In software development, a build is the process of converting source code files into a standalone software artifact that can be run on a computer, or the result of doing so" (Wikipedia 2020b).

Testing cross-platform apps is naturally more complex than the testing of an app that is targeting only one platform. A cross-platform app developed with React Native, for example, has to be tested on both iOS and Android emulators. An emulator makes it possible to test mobile apps on computers. Wikipedia (2020c) defines emulators as follows:

> In computing, an emulator is hardware or software that enables one computer system to behave like another computer system. An emulator typically enables the host system to run software or use peripheral devices designed for the guest system.

At the final stages of the development process, the app should also be tested on as many real devices as possible for each platform that the app targets to ensure that it functions properly. (Duffy 2018)

3.3 Cross-platform vs. native development

Before starting the development of the case app, I needed to decide if I wanted to develop a cross-platform or a native app. For me, it was important that I would be able to launch the app on multiple platforms while still building the app on my own. The mobile OS market in Finland is ¾ Android and ¼ iOS (Liu 2020). Because the case app is targeting such a niche segment, I felt strongly that I did not want to shrink an already small customer base by targeting only one app platform. The benefit of needing fewer programming languages and tools and being able to develop one app that can be published on multiple platforms, made it clear to me that cross-platform would be the better choice.

When reflecting on my skills it became even clearer. Through various university courses I had been able to familiarize myself with many different languages, frameworks, and technology tools, including JavaScript, React, Java, Android Studio etc. However, I had no experience in Swift or Objective-C. This meant that developing a native iOS app was out of the question as I would have needed to start learning from scratch. Then I had to consider if I wanted to develop an Android app using Android Studio and Java, or a cross-platform app using React Native, Xamarin, or another framework. Through my studies and personal development, I had become quite familiar with JavaScript and React, the language and library used in React Native, and C#, which is the language used for Xamarin, both of which are cross-platform frameworks.

3.4 Cross-platform technologies

Largely all cross-platform technologies strive to make the development of smartphone apps that target multiple platforms more efficient, but there are considerable differences in how different technological approaches achieve this goal. Next, we are going to look at the two general types of cross-platform apps. (Duffy 2018)

3.4.1 Hybrid cross-platform apps

Hybrid cross-platform apps are basically web apps built with HTML, CSS, and JavaScript, that are converted to native apps with a hybrid framework (see figure 5). The conversion creates an app that contains all resources needed for it to be downloaded and run locally on a smartphone. The downloaded app uses the WebView component of the smartphone's internet browser to run. This is different from how normal web apps work since they typically require an internet connection to work and are run on servers in the cloud, not locally. (Duffy 2018)

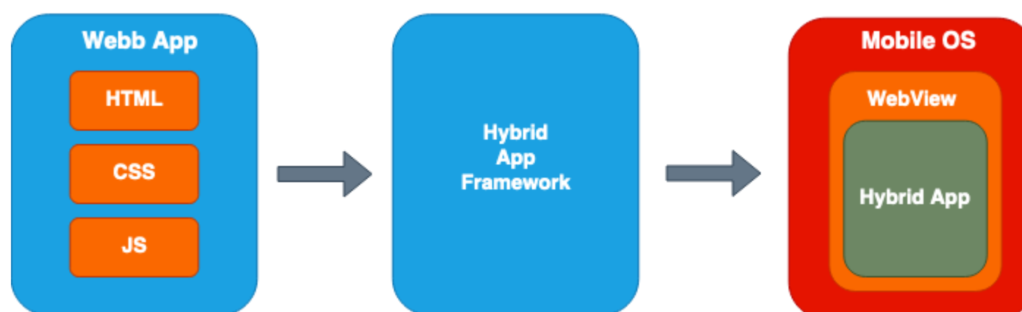

Figure 5. How hybrid cross-platform frameworks work (Duffy 2018).

Figure 6 provides a simplified overview of the architecture of a hybrid app. The OS loads the web app in the WebView component of the device, which is normally used for rendering resources on the internet. The WebView is a native component, so it has access to the OS native services, like sensors, graphics system, etc., and can receive input from the user. In cases where the WebView component is unable to access a specific native service, hybrid frameworks often supplement with plug-ins. (Duffy 2018)

Hansson & Vidhall (2016, p. 5) highlight the following regarding the performance and the look and feel of hybrid apps:

> Hybrid applications can reuse interface components, like a mobile web application, however it is hard to achieve an actual native feeling. Further drawbacks from using a web browser as core component in hybrid applications includes lower performance compared to native applications.

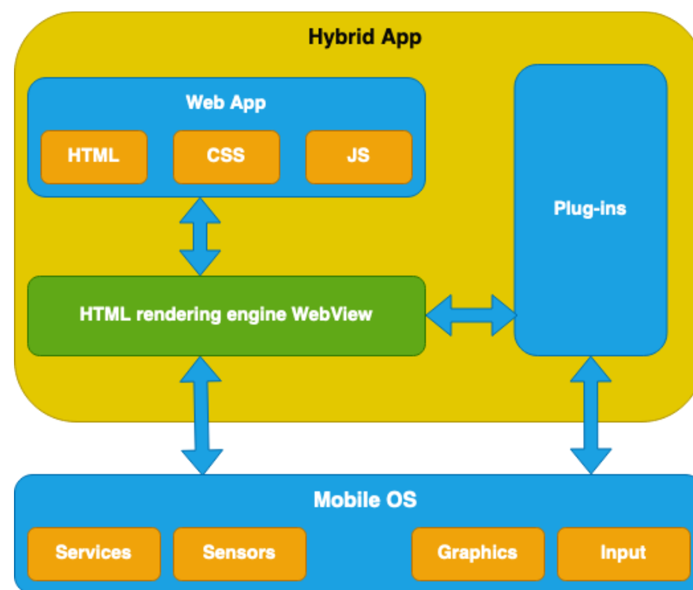

Figure 6. Hybrid app architecture (Duffy 2018).

3.4.2 Cross-platform native apps

Cross-platform native apps are written in a programming language that is not native to the OS of the smartphone. The cross-platform framework compiles and translates the written code to apps that can be downloaded and run natively on the device (see figure 7). The result is an app that is very similar to a native app written in the native language of the OS. (Duffy 2018)

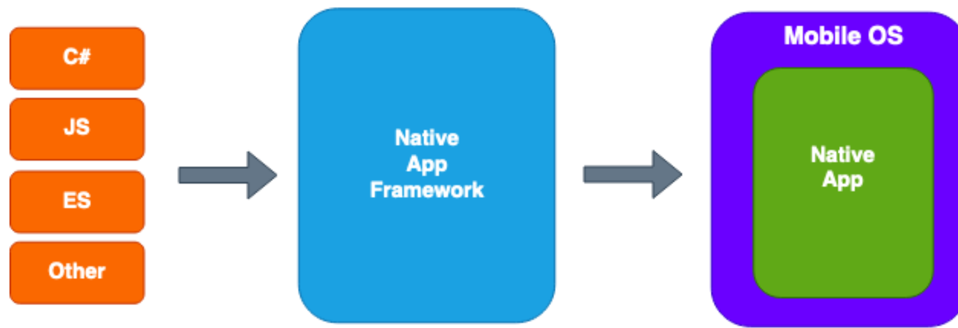

Figure 7. How cross-platform native frameworks work (Duffy 2018).

Similar to hybrid apps, the performance of cross-platform native apps can suffer because the apps are not utilizing the OS's native environment but accessing it through an additional software layer. (Hansson & Vidhall 2016, p. 6)

Because every OS is different, some OS-specific code is also required. If the cross-platform native app uses a lot of OS-specific features, it will reduce the amount of shared code across platforms and make the apps more distinctive. Writing OS-specific code also requires some special knowledge about the native API. (Hansson & Vidhall 2016, pp. 5-6)

From an architecture perspective, there are two types of cross-platform native apps. The first type has the same code for business logic, which might include making calculations or calling a database, for all platforms but separate code for the user interfaces (see figure 8). Xamarin is one example of such a framework. The second type shares the business logic and user interface for all platforms. React Native works this way. (Duffy 2018)

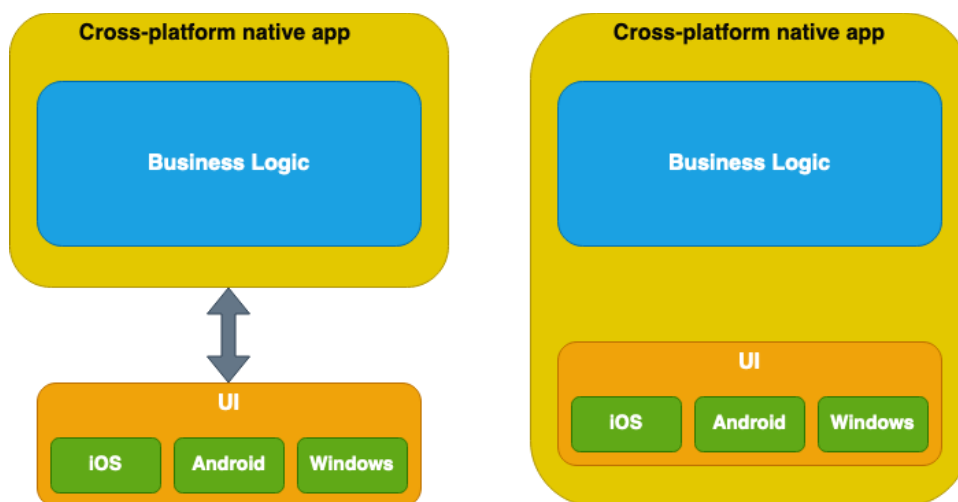

Figure 8. Two types of cross-platform native apps (Duffy 2018).

3.5 Why I chose React Native for developing the app

Hansson & Vidhall (2016, p. 2) emphasize that React Native reduces the needed technological know-how regarding iOS and Android to mostly React, and that saves time and resources. My research into React Native had shown mostly advantages and few disadvantages from the perspective of an individual developer with scarce resources. I had tried learning Xamarin a while ago but found it to be unfamiliar, which made the learning curve feel quite steep. Seeing as I have past experience with React, and I really liked working with the library, made it an easy choice.

3.5.1 A closer look at React Native

React Native is an open-source cross-platform framework created by Facebook in 2015 (React Native 2020). React Native uses the JavaScript programming language and the React web development library, which is meant for creating interactive web apps, to create native cross-platform apps for iOS and Android (Abbott & Djirdehh 2020, p. 9).

React Native apps are a little bit faster than hybrid cross-platform apps because the framework creates an app that is run natively on the device. This is different from hybrid cross-platform apps that run within the native web component of the device. React Native also comes with some very useful features. One of them is called "Hot Reload" and it allows developers to run the app while they are developing it and almost instantly see what they are implementing. When new functionality has been programmed, and the code file is saved, "Hot Reload" will rebuild the app and integrate the new functionality, which makes it quick and easy to see and test new functionality. The framework also makes it easy to integrate native code with React Native. If a certain component needs to be optimized, it can be written in the OS's native programming language. Thanks to the popular React web development library, React Native also has a large and active developer community that supports its ecosystem and development tools. (Duffy 2018)

Many big companies, for example, Tesla, Pinterest, Uber Eats, and Discord, use React Native (React Native 2020a). However, it is unclear if the framework can support very complex and ambitious apps. Peal (2019) recently reported how Airbnb is going to

transition away from React Native after working with the framework for many years. There are many reasons for this, both organizational and technological, but in a nutshell, they could no longer harness the strengths of the framework because they were developing so much OS-specific code. Peal (2019) explains further:

> Even though code in React Native features was almost entirely shared across platforms, only a small percentage of our app was React Native. In addition, large amounts of bridging infrastructure were required to enable product engineers to work effectively. As a result, we wound up supporting code on three platforms instead of two.

* * *

Choosing the technological tools to build with often comes down to a mixture of the client's needs, the requirements, and the preferences and know-how of the developers. If the requirements allow, as they do in this case, choosing a framework really is at the discretion of the developer. Now that we have decided what framework we want to use to develop the MVP app, we will need to drop down a level in detail and actually start using the framework, implementing the app, and making low-level decisions that directly affect how the app looks and feels. The next section is going to focus on the development of the app.

4 BUILDING THE MVP WITH REACT NATIVE

This section gives an overview of how the MVP was built and the technological decisions that were made in the process. It explains the functionality of the app in a broad sense and assumes some prior programming knowledge and familiarity with React and React Native.

First, we will look at requirements, tools, and installation and setup to let the reader know where the development process started. Then we will cover the main building blocks of the app in the order that they were programmed. Finally, we will look at the functionality that was added to improve the user experience of the app. The app development is largely based on the book Fullstack React Native by Abbott & Djirdehh (2020), and a few online resources, like Stack Overflow posts and Github repositories.

4.1 Preparation

4.1.1 Requirements

The requirements were approached in a humble manner because I was only a third-year IT engineering student when starting this project, and I had never built a smartphone app before. The goal was for the app to mimic the structure of the hunting exam and take inspiration from other interactive learning material. Essentially, this means that the app should have two different types of questions: text-based questions with multiple-choice answers, and image-based animal identification questions with multiple-choice answers. The animal identification questions can be split into two broad categories: birds and mammals, which is also what I chose to do in the app. To allow the users to learn, the app must tell them what went right and what went wrong. So, the app needed to have a results page that shows information about how the user performed.

4.1.2 Tools

On the hardware side, I used an iPhone 7 Plus and a 2019 MacBook Air with a 13.3-inch display. On the software side, I used Visual Studio Code as my integrated development environment (IDE). Visual Studio Code is a free open-source editor developed by Microsoft.

Developing an MVP is not costly, the only essential tools for writing code, publishing the app, searching for information, and learning are a decent laptop and internet connection. The biggest and only barrier to entry is time - the time it takes to learn to code and build something by utilizing existing technological tools and frameworks. Technological advancements have made computers more affordable and the internet more widely accessible. This means that anyone who has access to a computer and the internet and is willing to invest their time and energy can become an app developer. This is most likely one of the reasons why there are so many different apps available, and why competition among those apps is so fierce.

4.1.3 Installation and setup

4.1.3.1 Yarn

To install the necessary libraries and packages needed for developing the app, I chose to use the yarn package manager. Abbott & Djirdehh (2020, p. 20) recommend yarn as they have found it produces more consistent builds for React Native than npm, which is another common package manager. Abbott & Djirdehh (2020, p. 20) explain how yarn works:

> yarn is a JavaScript package manager – it automates the process of managing all the required dependencies from npm, an online repository of published JavaScript libraries and projects, in an application. This is done by defining all our dependencies in a single package.json file.

4.1.3.2 Expo

After I had installed yarn, I moved on to Expo. React Native documentation recommends the use of Expo when first getting familiar with the framework (Abbott & Djirdehh 2020, pp. 20-21). Expo is a platform that offers various tools that are designed to build fully functional React Native apps without having to write any native code. The Expo command line interface (CLI) makes it very easy to start a new React Native project, with one command, without spending time on build configuration. Expo also offers a smartphone app that makes it easy to run the app on a device while programming. This made it quick and easy to see and test the app in small increments as I was programming it.

Expo has two disadvantages. First, if the project requires native code, one must eject from Expo to implement it. Second, if Expo is used for building the app, Expo will include every device API in the final build of the app, regardless if it is used or not, resulting in the size of the final app build being larger than it needs to be (Abbott & Djirdehh 2020, 672).

Expo offers one great advantage that comes into play after initial development has been completed and the first version of the app has been published. Publishing the app with Expo makes it possible to update it “over-the-air” (OTA) (Abbott & Djirdehh 2020, p. 850). Abbott & Djirdehh explain: “By default, applications published with Expo will always check for updates when launched”. If a new version of the app has been published, it will automatically be fetched and loaded the next time the user restarts the app. This makes it possible to make small changes and fix certain errors without submitting a new

build to the app stores and going through the iOS or Android app review process. However, OTA updates only work when JavaScript code is modified (Abbott & Djirdehh 2020, p. 896).

4.1.3.3 Babel

React Native uses Babel as it is JavaScript compiler (Abbott & Djirdehh 2020, p. 18). Babel compiles code written in newer JavaScript versions so that it is backwards compatible and works on machines that use older JavaScript versions (Wikipedia 2020d). This allowed me to write code according to the latest JavaScript version syntax and use the newest functionality that the programming language has to offer.

4.1.4 The beginning

To get myself warmed up and to learn about the different features that the React Native framework has to offer. I first went through some of the example projects in the book by Abbott & Djirdehh (2020) that I had bought. Then I went for some walks and had some coffee and visualized what the app might look like in my head. I had been thinking about the app for weeks before I actually started developing it.

Developing a cross-platform smartphone app was new to me, and that made me unsure if I should start by writing down requirements, creating a mockup of the app in a graphics program, or jump straight into coding. I had recently read some of Sommerville's (2016) and Stephens (2015) Software Development theory, but it felt overly complex for my type of app. Abbott & Djirdehh (2020, p. 111) introduced a seven-step process for building React Native apps that felt more approachable. They advised to start by first breaking the app into components and then building a static version of the app. Somewhat influenced by this I decided to start. I initiated a new React Native project with Expo, which provided me with a blank screen. Then, I decided I would start by building a static menu, with the first thing being the button components. The development process was very informal and incremental as I moved from one component to another slowly building out the app, always keeping the MVP approach at the back of my mind when making any decisions.

4.2 Development

4.2.1 Buttons

Figure 9. Screenshot of app buttons.

In React Native, it is possible to create buttons in a few different ways. Abbott & Djirdehh (2020) use Button, TouchableHighlight, and TouchableOpacity core components in their example projects. You can create "buttons" with all of them with a slightly different syntax. I decided to use TouchableOpacity because the component incorporated a fade effect when interacted with. The component I choose to create the buttons with did not seem like a very critical decision, so I did not spend much time comparing the three alternatives. I did try to use Button but for some reason found it hard to style it consistently, so it looks the same on Android and iOS.

On the React Native website, TouchableOpacity is described as "a wrapper for making views respond properly to touches. On press down, the opacity of the wrapped view is decreased, dimming it" (React Native 2020b).

All React Native core components that are used need to be imported at the top of the file. This includes things like View, Text, StyleSheet, and TouchableOpacity, which were used a lot.

```
import { SafeAreaView, Text, StyleSheet, TouchableOpacity, Modal } from 'react-native'
```

Figure 10. Example of React Native core component importing.

After I had my first TouchableOpacity component created, I could then copy and paste the component to create two more buttons, swapping out the content of the child text element for the other two (see figure 9). Next, I had to think about styling the buttons.

4.2.2 Styling the components

```
const styles = StyleSheet.create({
    button: {
        alignItems: "center",
        backgroundColor: "#248f24",
        padding: 20,
        color: 'white',
        borderRadius: 25,
        margin: 20,
        shadowColor: 'black',
        shadowOffset: { width: 0, height: 3 },
        shadowOpacity: 0.5,
        shadowRadius: 5,
        elevation: 5
    },
    buttonText: {
        color: 'white',
        fontFamily: 'Verdana',
        fontSize: 30,
    },
})
```

Figure 11. Screenshot of button styling.

Not all components in React Native accept styling, but most do (Abbott & Djirdehh 2020, pp. 37-38). Styling in React Native is very similar to styling with Cascading Style Sheets (CSS) and easy to learn for someone who has done some web development (see figure 10). However, Abbott & Djirdehh point out that even though the syntax looks similar to CSS, it is not CSS.

To layout and align items, React Native uses Flexbox. The column layout is the default for Flexbox, so the buttons were automatically stacked vertically (see figure 9). Styling is a very complex topic in itself, so I won't go deeper into it as it would become too detailed.

4.2.3 Screens and navigation

Once I had the buttons done and styled, I had to create the different screens that the buttons would open up when they were tapped. Figure 11 gives an overview of what

Figure 12. A few screenshots of screens from the app. From left to right, Menu, ImageQuestions, and ResultsView.

the screens in the actual app look like. I will cover the screens for the two question categories, and the results list in the upcoming chapters of this section. The final application

contains four different screens: Menu, TheoryQuestions, ImageQuestions, and ResultsView. TheoryQuestions and ImageQuestions are basically the same screen with minor differences.

4.2.3.1 Modal

A screen combines different components and a user can only see one at a time. Abbott & Djirdehh (2020) present two options for navigating a user between screens. You can use a navigation library or the Modal component. Abbott & Djirdehh (2020, p. 284) recommend the Modal component for simple apps, so I decided to go with their recommendation, with the option of changing to a navigation library if I felt Modal was inadequate. I did not expect the app to have many screens so I felt Modal would be good enough.

The Modal component allowed me to open and close different screens by linking them to buttons. React Native describes Modal as "[…] a basic way to present content above an enclosing view" (React Native 2020c). When a button in the menu screen is tapped, it will open up a new screen in front of the menu screen. Figure 12 is a sequence diagram of the app. It illustrates how the user can move between the different screens. Next, I needed to figure out a way for the user to close the screen, mid quiz, if they wanted to.

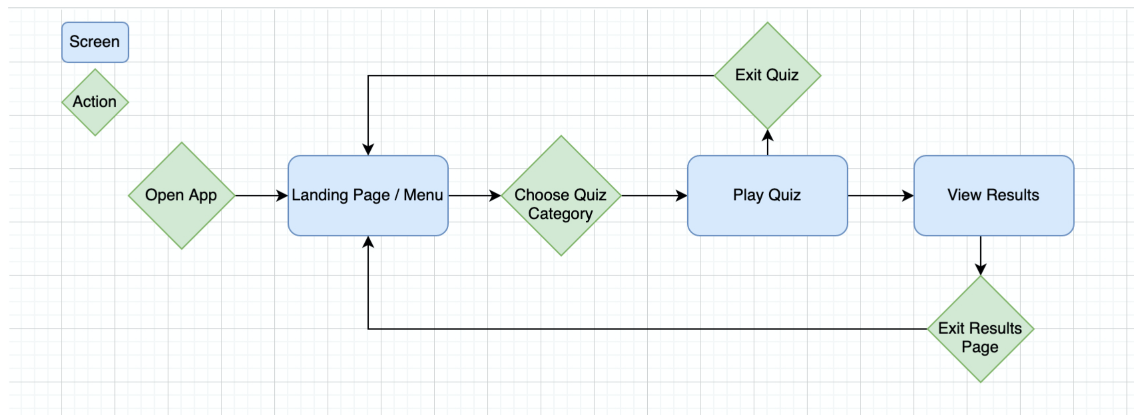

Figure 13. Sequence diagram of the app navigation flow.

4.2.3.2 NavigationBar

The NavigationBar is a component that Abbott & Djirdehh (2020) introduced in one of their example projects. I felt it would fit well in this application, so I decided to implement it. The NavigationBar contains a title and a button that can be used to close the current screen and get back to the menu screen (see figures 13 and 14). In the end, the NavigationBar component was used on three screens; TheoryQuestions, ImageQuestions, and ResultsView.

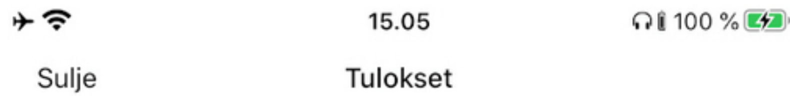

Figure 14. Image of NavigationBar component.

4.2.4 Quiz logic and questions

After I had figured out how to enter and exit a question category, I then needed to build out the question screens, figure out how the questions should work, and what the underlying logic would be for checking if an answer was right or wrong.

I was not sure how I would build out the quiz logic, so I did some googling and found a quiz app example by Spencer Carli (2020) on Github. I investigated the code and decided to implement a part of his example, with some small modifications. This example helped me decide on the data structure for the questions. It also allowed me to add a counter at the bottom of the question screens to give the user an idea of how many questions had been answered and how many were remaining.

```
<ResultsView
  onClose={this.closeResults}
  wrongAnswers={wrongAnswers}
  correctCount={correctCount}
  totalCount={totalCount}
 />
```

Figure 15. ResultsView component and its props.

Once a category of questions is finished, the Quiz component sends four properties to the ResultsView screen. One function, one object with the wrong answers, and two variables. Then, the ResultsView screen opens up on top of the current question screen. When the ResultsView screen is closed, it also closes the underlying question screen, returning the user to the Menu screen. This happens so fast that the user does not have time to notice that two Modals are closed by the tap of one button.

4.2.4.1 Text-based questions

```
{
      question: "Mille seuraavista linnuista rakenne
      answers: [
        { id: "1", text: "Telkkä", correct: true },
        { id: "2", text: "Sinisorsa"},
        { id: "3", text: "Metso" },
      ]
  }
```

Figure 16. Data structure of text-based question.

The text-based questions were stored in an array of objects (see figure 16) that were then made available to the rest of the program by exporting. The questions are imported to the Menu screen and from there passed down as properties to

the other screens and components. The Quiz component goes through the question object, rendering each question one by one, registering which questions are answered incorrectly, and counting how many are answered correctly.

4.2.4.2 Image-based questions

```
{
        question: "Mikä laji on kuvassa?",
        answers: [
            { id: "1", text: "Hirvi", correct: true },
            { id: "2", text: "Metsäkauris" },
            { id: "3", text: "Kuusipeura"},
        ],
        source: { uri: "https://metsastysvisa.s3.eu-nort
```

Figure 17. Data structure of image-based question.

The image-based questions work the same way as the text-based questions, although they have a slightly different object structure due to the object that is stored as the value for the source key. The image-based questions and the text-based questions share most of the same code, but I use conditional rendering to check if the question object has a source key, and if it does, I render an image between the question and the answers.

The images are hosted on an Amazon Web Service (AWS) S3 bucket. They have been resized to make them smaller which will allow them to load quicker. Originally, I was hoping to store all of the images in the assets of the application, which means they would be available offline, and would be downloaded to the smartphone when the app is installed. However, I did not find an easy solution for this that would have worked together with the quiz logic and the shuffle algorithm, which will be discussed in chapter 4.2.4.3, that I use to shuffle the questions. Finland has great 3G and 4G networks across the whole country, so I did not see any reason why the images could not be fetched from a cloud server. Therefore, I decided to move forward with using images hosted on a cloud server and try to figure out how to make the app work offline later on.

4.2.4.3 Shuffle

To avoid the user memorizing the answers to the questions due to their order, I needed to come up with a way to shuffle the array of objects before I pass it down to the Quiz component, which takes care of the rendering. Writing an algorithm that shuffles an array of objects is not something I was sure I would be able to do, and even if I did, I would still need to compare it with other algorithms online to make sure I did not make any mistakes and that my solution was appropriate. I also felt this is a problem that must have been solved before. To avoid wasting time, I decided to do some googling and then found

a solution on Stack Overflow that was approved by 1611 other users (Stack Overflow 2010). The algorithm presented in the solution was the Fisher-Yates algorithm. I implemented the algorithm in my code, did some testing, and it worked well.

4.2.5 ResultsList

All three question categories have over 30 different questions and one of them has close to 50. It's possible that a user answers all questions incorrectly, which means they then need to be able to review the correct answers to all questions. Abbott & Djirdehh (2020) introduced two list components in their book, ScrollView and FlatList. Abbott & Djirdehh (2020, p. 274) explain that ScrollView works best for smaller lists with less than 20 items as it renders every item in the list even if most of them are outside of the screen. FlatList is a better solution if the list might have more than 20 items since it does not render all of the items at once. FlatList leaves some of the items that are outside of the screen unloaded until the user swipes closer to those items (Abbott & Djirdehh 2020, pp. 250-251). Since the ResultsList component might have up to 50 items with images. I decided to use FlatList for the ResultList component on the ResultView screen. Figure 12 illustrates what the ResultsView screen looks like.

4.2.6 State

In React, states are used to keep track of changes in components, and whenever a value stored in a components state is changed, the component re-renders (React s. a.). In the case app I have state in four places: in the Menu screen, the Quiz component, the Status component, and the Card component. In their seven-step process, Abbott & Djirdehh (2020) recommend that developers decide where state should be early on in the development process. Due to lack of experience, I chose not to follow their advice. Instead, I developed the app incrementally and added state to a component whenever I needed it. The state of the quiz component is the most complex. It contains Booleans, some numbers, and an array that makes the quiz logic work, and it collects data to be passed to the ResultsView screen. The state of the Menu screen is very simple and only contains a Boolean and a number that helps manage the screens or Modal components. The purpose of the state in the card component is to help with the loading indicator, which is described in detail in section 4.2.7.2. The Status component handles the network connectivity indicator and its state has one Boolean value for that purpose.

4.2.7 User experience-enhancing functionality

This section will discuss some additional features that I considered low-hanging-fruit, meaning I was familiar with them from Abbott & Djirdehh's (2020) examples and, therefore, knew how they could be implemented to improve the user experience.

4.2.7.1 Network connection alert

An internet connection is necessary for the app to work because the MVP fetches the images for the image-based questions from a server in the cloud. People often use their phones in areas with spotty service, so I wanted to create an alert that warns the user if there is no internet connection, in which case the image-based questions would not work.

To know if the device has an internet connection, I needed to use the NetInfo API, which is a React Native core API, that allowed me to easily access lower-level native APIs for iOS and Android. When calling the NetInfo API, the function returns a Boolean that tells us if the device is connected to the internet or not. You can attach an event listener to NetInfo so that we can get continuous updates regarding the network connection. I implemented the network connectivity indicator in a component called Status. I then used this component on all five screens of the app. Figure 18 illustrates what the network connectivity alert looks like when triggered, the example in the image was triggered manually and not by a real network issue, hence the full bars. (Abbott & Djirdehh 2020, pp. 307-322)

Figure 18. Network connection notification.

4.2.7.2 Loading

Before a user's device loses its internet connection, the connection might just get very slow. In anticipation of this, I wanted to implement a loading indicator for the image-

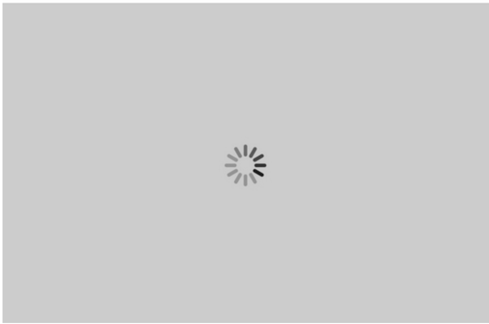

Figure 19. Image of ActivityIndicator.

based questions and the ResultsList, if the list was showing image questions. The ActivityIndicator, shown in figure 19, appears when an image is loading. The component is triggered by a Boolean in the state of the parent component. To let the ActivityIndicator know when an image has finished loading, we use the onload property of the image component. The ActivityIndicator is shown until the onload property is triggered to indicate that the asset has loaded completely.

4.2.7.3 Animations

I wanted to enhance the Menu screen for when the user opens up the app, so I looked into animations. Abbott & Djirdehh (2020) introduced two React Native animation APIs. One of them is called LayoutAnimation and the other Animated. I tried out both of them but decided to do the animations with Animated as I felt I had a better understanding of how it worked. Also, Abbott & Djirdehh did not recommend LayoutAnimation for moving around objects on the screen. One of the animations I wanted to do was to move the logo from the middle of the screen to the top, and then, when the logo was out of the way I would fade in the menu buttons from the white background. The Animated API made it possible to create these animations and to adjust durations and delays for the animations as needed.

4.2.7.4 Splash screen

Expo has a lot of helpful documentation and guides about creating and publishing apps to the app stores. One of the things they mention there is the splash screen (Expo s.a. a). The splash screen comes up after the user has tapped the app icon and stays visible while the app is loading. Reading through the Expo documentation, it seemed like something worth implementing as all professional apps have splash screens. I bought an image from a stock image site and then added my app logo to it. I then placed the splash screen image in the assets folder and specified the path for the image in the correct property field in the app.json file. The app.json file has a special purpose for the app. Expo documentation explains that "app.json is your go-to place for configuring parts of your app that do not belong in code […]" (Expo s.a. b). "app.json configures many things, from your app name

to icon to splash screen and even deep linking scheme and API keys to use for some services […]" (Expo s.a. b).

4.2.7.5 App Icon

To make sure the app looks professional it has to have an app icon. This is the icon the user will see on their device screen and then tap with their finger when they want to launch the app.

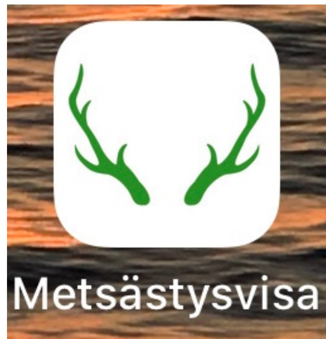

Figure 20. Image of app icon on iOS.

Expo provides great documentation on creating an app icon for iOS and Android devices (Expo s.a. c). The documentation explains that it is possible to use the same image for the app icons on both platforms. iOS requires the app icon to be a .png file with a size of exactly 1024x1024, the same size image will also work on Android. I created the icon image and placed it in the assets folder of the app. Then I specified the path to the image in the correct property field in the app.json file. You only need to specify it in one place for Expo to then use the same image on both platforms.

4.3 Testing

Throughout the development of the app, I was continuously testing the app on my iPhone 7 plus, but once I got to the stage where I felt I had done everything that was required, I needed to test it on devices with different screen sizes. Abbott & Djirdehh (2020) do not really mention this in their book, so I was not sure if the layout of the app would work on all screen sizes since I had been adopting many of their examples in the app. I feared this part of the development process might create problems because of how big the differences are between the really big smartphones and the small ones.

I decided I would test the app on three different screen sizes for both iOS and Android by using their respective emulators. I chose a big and a small screen and one in between. Figure 21 displays the small screen sizes that I chose, the devices from left to right are the iPhone SE with a 4.7-inch screen, and the Nexus S with a 4-inch screen size. With small adjustments, I was able to make the app look good on all devices that I tested on.

However, even though the app might look good on an emulator it does not guarantee that the final build of the app that will be distributed in the app stores will look the same.

Figure 21. Image of iPhone SE and Nexus S emulators

With React Native and Flexbox, it is straightforward to build responsive layouts. Abbott & Djirdehh (2020, p. 210) explain that:

> The flex style attribute gives us the ability to define layouts that can expand and shrink automatically based on screen size. The flex value is a number that represents the ratio of space that a component should take up, relative to its siblings.

Adopting many of the examples by Abbott & Djirdehh (2020), I naturally used the flex style attribute throughout my app. As a result of this, the app looked okay enough, with elements a tiny bit too big on smaller screen sizes and a tiny bit too small on larger screens, but overall still very functional. Because of this, I felt it was worth improving the responsiveness of the app at a later stage if it seemed valuable to its users.

4.4 Code sharing between platforms

One of the main benefits of the React Native framework is the possibility of creating an app with one codebase that can be distributed on the two largest smartphone platforms. In the end, the case app was able to take full advantage of this benefit as almost all of the code was shared between the platforms, the small amount of platform-specific code that was incorporated was UI related. It might very well be possible to avoid any platform-specific code in the case app if it was prioritized.

4.5 Reflecting on my development process

Looking back at the activities of my app development process, one could definitely characterize them as informal and incremental. Requirements engineering entailed me combining my short list of wants and needs for the app's functionality with observations of hunting exam related online learning environments and quiz-style smartphone apps, and

then constructing a mental image of what I wanted to build. This was a very quick process, and I felt I was ready to start designing and implementing the app immediately.

I started design and implementation, based on my mental image, from the first component I felt the user would interact with. Then I iterated my way through the project, one component at a time. Visualizing what the component would look like, thinking about the underlying system functionality, and looking at the examples in the React Native book I had bought. After a component was complete, I then tested it and styled it. My incremental and informal approach worked out well because the list of requirements was not very long, I was developing the app alone, and the app only contained the most essential functionality, as an MVP should. However, in regard to the app working offline, I think my lack of formal requirement specification had a negative impact. I think I might have been able to make the app work fully offline if I had prioritized it as a must-have requirement from the start and not a nice-to-have requirement as I did now. Because I did not prioritize it, I was not able to implement it at the moment when I felt I would have liked to because it did not fit easily into the functioning code I had already built. Reflecting on the process overall, I think there were four main things that significantly helped in successfully completing the first version of the app:

1. I'm happy I took my time to get properly familiar with examples in the book Fullstack React Native by Abbott & Djirdehh (2020), even though I was familiar with React and JavaScript, instead of rushing into development. Completing many of the example projects helped me to get familiar with the framework while building up my confidence before I started working on my own app. The example projects used many different React Native components and whenever I was about to start the development of a new component in my app, I could go back to the book to get inspiration and see all the different ways it could be done. Had I started developing the app right away without getting familiar with the framework, there would have been a much higher risk of problems and churn.
2. Before I started the project, I considered the MVP approach to be simple and practical. I then got even more excited about the approach as I read about Agile and discovered that that approach also uses a very similar method to gather early feedback to guide development and ensure that the product meets the needs of the customer/end-user. The aforementioned approaches helped me keep my requirements for the initial version of the app humble and supported me in my decision

making when faced with alternatives or problems during design and implementation. Deciding not to implement all imaginable features in the first version of the app, allowed me to keep the app simple and the goals attainable, which I think is important when working alone and without much experience with the technology being used.

3. React Native is a great fit for a developer with prior knowledge of JavaScript, React, and CSS. This made it very familiar and easy to develop functionality and style it. I also felt React fit really well with the incremental development approach I used as it is a component-based library.
4. Expo made the app development straightforward in a few different ways. First, Expo makes it simple to start a React Native project without spending much time on configuration. Second, the hot reloading feature made the development of the app very convenient, because it allowed me to quickly see and test the functionality I was building during design and development. This made coding more fun, helped me iterate my way forward while coding components, and encouraged me to test and locate bugs early on.

* * *

With the development of the MVP finished, the next steps would be to launch the app in the app stores. The development of the app would then continue based on user feedback if there are enough active users to justify its existence. But, the publishing of the app to the app stores and the its post-launch evolution is outside the scope of this thesis and will, therefore, not be covered in the upcoming section. Instead, I will conclude the thesis by discussing what I learned while creating the case app and writing this thesis. I will also discuss the research questions and examine how well I was able to answer them. Finally, I will evaluate the thesis as a whole in order to identify areas of improvement that might be of help to other future engineers that are developing and researching cross-platform apps.

5 CONCLUSION

This project has shown that it is very possible for an inexperienced sole developer to build a low-cost app that can be released on multiple smartphone platforms and potentially

reach billions of consumers. Making it possible for even niche app ides to have a sound business case and be potentially profitable. In addition, it demonstrates that improved technological tools, like cross-platform frameworks, have made app development increasingly easier and almost anyone with some thought and effort can develop an app. It also demonstrates that the skills needed for app development can be easily learned and that there are no real barriers to entry for smartphone app development.

After working with React Native throughout this project, I have grown to like the framework a lot and am thoroughly impressed by it. I think the framework is really successful in making developers more productive by allowing someone with prior knowledge of web development with React to leverage their skills to develop apps for multiple platforms, without sacrificing the quality of the app. The case app shares almost all code between platforms, the small amount of platform-specific code that was incorporated was UI related. This demonstrates that MVP development with React can be really practical. This is especially valuable for entrepreneurs and small businesses that have scarce resources and really need to maximize their time, effort, and money. It provides an opportunity for them to explore new ideas on mobile that they might have thought previously unfeasible, because of the amount of knowledge needed to create an app for several platforms.

Now that I have firsthand experience from the MVP approach, I can say it is a powerful concept, and especially useful for developers who are at the start of their careers and hope to build production level apps for their portfolio, or small teams who are looking to try out a new app related business idea. I think it is most important to keep the requirements simple and reasonable for the first versions of the app, in order to get to a functioning prototype quickly and build up momentum for the project.

Whenever I am learning something new, I find it increasingly important to first find good learning resources. In my early research, I began looking for theory to support me in my development of the MVP app. The software development theory that I studied from Sommerville (2016) and Stephens (2015) was comprehensive and of high-quality, but I had a hard time relating most of the theory to the issues I was pondering around my project, as it was clearly meant for projects of greater scale and complexity. I found myself spending a lot more time with the book from Abbott & Djirdehh (2020), which was less theoretical and more like a cookbook, containing many code examples and more detailed

descriptions of what components to use and why. For anyone considering developing a simple MVP app, alone or in a small team, I would recommend getting familiar with software development theory and incremental development models, but more important, I think, would be to find a good software development cookbook that is closely related to the technology being used in the project, and really studying it thoroughly, in order to leverage the strengths of the technology and easily use the components it offers. I feel the importance of software development theory and a proper process, meaning, for example, a more formal and documented waterfall approach or an informal but more structured agile approach, will increase as the scale of a project grows and more people get involved. However, as long as the project has the scale of a prototype, I think it is more valuable to focus on the development of the app and less on documentation. As one can easily build something with a short list of requirements and an incremental development process. Incremental development being the key in this case as it takes the project through multiple loops of design and development. Allowing developers to continuously adjust the requirements of the project throughout the development process.

As I'm looking to take my next step in my professional journey as a software developer, I'm often surprised by the amount of name dropping of different programming languages, frameworks, and libraries in job advertisements, even for entry-level positions. As demonstrated in this thesis, React Native at least tries to counter this as it extends the React library to mobile development for multiple platforms, which I find is really valuable for aspiring developers who might feel overwhelmed by the number of programming languages they can choose to learn, or the thought of having to choose which part of the stack they want to specialize in before they even know what it entails. With JavaScript, React, and React Native one can build for web and mobile, one can build frontends and backends, and this makes it a great bundle of technology to choose, with confidence, to kick-off ones learning as a hobbyist or someone who is aspiring to become a professional developer because the technologies cover the whole stack and make it possible to build almost anything.

Reflecting on this project as a whole I’m quite satisfied with the result as it gave me time to explore and pursue things I have been interested in for a long time. In addition, I have now built a simple production-level app that I feel will be of interest to its intended niche market and perhaps my future employer. Ever since I started studying information

technology engineering at Arcada, I had a goal of learning how to build apps that could be released to real customers, and this helped me achieve that. I hope this thesis makes the process of creating a smartphone app less intimidating for young and aspiring engineers, and that the results would encourage them not to dismiss the app ideas that they might have, but instead simplify, build, and publish them, so others can decide their value and give feedback to help improve them.

My recommendation for further research is around MVP smartphone app development. Throughout this project I was looking for MVP-style software development theory and I was unable to find any from the resources available to use via Arcada UAS. I think there is room for additional software development theory that focuses on the development of MVPs, between the thorough, high-level, and sometimes complex software process theory and the very detailed software development cookbooks. This is something I would have greatly enjoyed during this project.

REFERENCES

Abbott, D. & Djirdehh, H. (2020). *Fullstack React Native.* [E-book] Leanpub. Available at: https://www.newline.co/fullstack-react-native/ [17.6.2020]

Android (s. a.). *Design & Quality.* [Web site] Available at: https://developer.android.com/design [6.11.2020]

App Annie (2020). *State of Mobile 2020.* [pdf] App Annie. Available at: https://www.appannie.com/en/go/state-of-mobile-2020/ [20.4.2020]

Apple (s. a.). Design. [Web site] Apple. Available at: https://developer.apple.com/design/ [6.11.2020]

Baker, J.C. (2018). *Getting to Market with Your MVP: How to Achieve Small Business and Entrepreneur Success.* [E-book] Business Expert Press. Available at: https://www.perlego.com [21.8.2020]

Bjørn-Hansen, A., Grønli, T., Ghinea, G. & Alouneh, S. (2019). An Empirical Study of Cross-Platform Mobile Development in Industry [Journal Article] *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1-12. Available at: https://www.hindawi.com/journals/wcmc/2019/5743892/ [7.6.2020]

Chen, B. X. (2010). Apple Registers Trademark for 'There's an App for That'. [Online article] *Wired.* Available at: https://www.wired.com/2010/10/app-for-that/ [19.8.2020]

Clark, J.F. (2012). *History of Mobile Applications*. [pdf] MAS 490: Theory and Practice of Mobile Applications. University of Kentucky. Lexington 4.4.2012. Available at: http://www.uky.edu/~jclark/mas490apps/History%20of%20Mobile%20Apps.pdf [20.4.2020]

Carli, S. (2020). react-native-quiz. [Github repository] Available at: https://github.com/ReactNativeSchool/react-native-quiz [5.7.2020]

Duffy, T. (2018) *Choosing a Cross-Platform Development Tool: Cordova, Ionic, React Native, Titanium, and Xamarin,* (2018). [Online course] Linkedin.com, 8.5.2018. Available at: https://www.linkedin.com/learning/choosing-a-cross-platform-development-tool-cordova-ionic-react-native-titanium-and-xamarin/what-you-should-know?u=56747801 [7.8.2020]

Expo (s.a. a). *Create a splash screen.* [Web site] Expo. Available at: https://docs.expo.io/guides/splash-screens/ [9.9.2020]

Expo (s.a. b). *Configuration with app.json / app.config.js.* [Web site] Expo. Available at: https://docs.expo.io/workflow/configuration/ [9.9.2020]

Expo (s.a. c). *App Icons.* [Web site] Expo. Available at: https://docs.expo.io/guides/app-icons/ [9.9.2020]

Hansson, N. & Vidhall, T. (2016). *Effects on performance and usability for cross-platform application development using React Native.* Dissertation. Linköping: Linköping University

McWherter, J. and Gowell, S. (2012). *Professional Mobile Application Development.* [E-book] Wiley. Available at: https://www.perlego.com [6.11.2020]

Mikkola, M. (2018). Metsästäjätutkinto uudistui – uusia metsästäjiä yli 6 000. [Online article] *Riistan Vuoksi.* Available at: https://www.riistanvuoksijulkaisu.fi/artikkelit/riista-tieto/metsastajatutkinto-uudistui-uusia-metsastajia-yli-6-000.html [20.8.2020]

Mushtaq, Z., Kirmani, M., Saif, S.M., & Wahid, A., (2016). Mobile Application Development: Issues and Challenges. [Journal article] *International Research Journal of Engineering and Technology,* vol. 3, pp. 1096-1099 Available at: https://www.researchgate.net/publication/327154905_Mobile_Application_Development_Issues_and_Challenges [6.11.2020]

Manchanda, A. (2019). *Where Do Cross-Platform App Frameworks Stand in 2020?* [Online article] Net Solutions. Available at: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/ [10.8.2020]

PC Mag (s. a.) *Mobile Platform.* [Web site] PC Mag. Available at: https://www.pcmag.com/encyclopedia/term/mobile-platform [6.11.2020]

Peal, G. (2019). *Sunsetting React Native.* [Weblog] Medium, 19.6. Available at: https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a [11.8.2020]

Poppendieck, M. and Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash.* [E-book] Addison-Wesley Professional, Available at: https://www.perlego.com [8.5.2020]

React (s. a.). *Component state.* [Web site] React. Available at: https://reactjs.org/docs/faq-state.html#gatsby-focus-wrapper [10.11.2020]

React Native (s. a. a). *React Native Learn once, write anywhere.* [Web site] React Native. Available at: https://reactnative.dev/ [11.8.2020]

React Native (2020b). *TouchableOpacity.* [Web site] React Native. Available at: https://reactnative.dev/docs/touchableopacity.html [4.9.2020]

React Native (2020c). *Modal.* [Web site] React Native. Available at: https://reactnative.dev/docs/modal#docsNav [4.9.2020]

Ries, E. (2011). *The Lean Startup: How Constant Innovation Creates Radically Successful Businesses.* St Ives: Portfolio Penguin.

Riistainfo.fi (s.a. a). *Suomalainen metsästys.* [Web site] Riistainfo.fi. Available at: https://www.riistainfo.fi/suomalainen-metsastys/ [19.8.2020]

Riistainfo.fi (s.a. b). *Metsästäjätutkinto.* [Web site] Riistainfo.fi. Available at: https://www.riistainfo.fi/uusi-metsastaja/metsastajatutkinto/ [20.8.2020]

Stack Overflow (2010). *How to randomize (shuffle) a javascript array.* [Web site] Stack Overflow. Available at: https://stackoverflow.com/questions/2450954/how-to-randomize-shuffle-a-javascript-array?page=1&tab=votes#tab-top [9.7.2020]

O'Dea, S. (2020). Market share of mobile operating systems worldwide 2012-2019. [Online article] *Statista.* Available at: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ [7.8.2020]

Liu, S. (2020). Leading mobile operating systems in Finland as of June 2020, by market share. [Online article] *Statista.* Available at: https://www.statista.com/statistics/623924/most-popular-mobile-operating-systems-in-finland/ [9.11.2020]

Stephens, R. (2015). *Beginning Software Engineering.* [E-book] Wiley. Available at: https://www.perlego.com [6.8.2020]

Strain, M. (2015). 1983 to today: a history of mobile apps. [Online article] *The Guardian.* Available at: https://www.theguardian.com/media-network/2015/feb/13/history-mobile-apps-future-interactive-timeline [21.4.2020]

Sommerville, I. (2016). *Software Engineering, Global Edition. 10th ed.* [E-book] Pearson Education Limited. Available at: https://www.perlego.com [6.8.2020]

Wikipedia (2020a). *iPhone.* [Web site] Wikipedia. Available at: https://en.wikipedia.org/wiki/IPhone [6.8.2020]

Wikipedia (2020b). *Software Build.* [Web site] Wikipedia. Available at: https://en.wikipedia.org/wiki/Software_build [12.8.2020]

Wikipedia (2020c). *Emulator.* [Web site] Wikipedia. Available at: https://en.wikipedia.org/wiki/Emulator [12.8.2020]

Wikipedia (2020d). *Babel (transcompiler).* [Web site] Wikipedia. Available at: https://en.wikipedia.org/wiki/Babel_(transcompiler) [9.7.2020]

APPENDIX

6 SWEDISH SUMMARY

Om det finns en enkel och vardaglig uppgift som kan lösas med hjälp av teknologi, finns det säkert också en mobilapplikation som försöker göra det. Under senaste årtiondet har mobilapplikationsmarknadens värde exploderat till flera hundra miljarder medan konsumenter blivit allt vanare att söka efter applikationer för även nischbehov, vilket fortsättningsvis ökar efterfrågan. En sådan nisch kunde till exempel vara en applikation som hjälper blivande jägare förbereda sig för finska jägarexamen. Omkring 6000 personer klarar jägarexamen årligen. Men, för att affärsfallet för en nischapplikation med en så liten målgrupp skall vara vettig så måste utvecklingen vara billig och enkel.

Det finns digitalt inlärningsmaterial för jägarexamen på nätet men inte i formen av en mobilapplikation. Då jag höll på att förbereda mig för jägarexamen så sökte jag efter en mobilapplikation där jag kunde öva på inlärningsmaterialet för examen, men tyvärr hittade jag ingen. Detta enkla men ouppfyllda behovet fick mig att besluta att jag skulle utveckla en applikation för detta syfte som en del av mitt examensarbete.

I dagens läge är mobilmarknaden, som också inkluderar mobilapplikationsmarknaden, dominerad av två aktörer och sina mobiloperativsystem, Apple med iOS och Google med Android. Dessa mobiloperativsystem har sina egna utvecklingsekosystem, med skilda programmeringsspråk och utvecklingsverktyg, vilket betyder att man måste kunna programmera en applikation på två olika sätt om man vill lansera den på både iOS och Android plattformarna. Att programmera en applikation med samma affärslogik på två olika sätt kräver både tid och kunnande, vilket gör utvecklingsprocessen mycket dyr. Men, cross-platform-amverk erbjuder en lösning. Med cross-platform-ramverk kan man programmera en applikation som sedan med hjälp av ramverket översätts så att den fungerar på flera plattformar. Cross-platform-ramverk gör det billigare och enklare att utveckla mobilapplikationer för flera plattformar, och detta kan ha stor betydelse för småföretag och ensamma entreprenörer med knappa resurser.

Målet med detta examensarbete var att utveckla en mobilapplikation genom att utnyttja ett cross-platform-ramverk tillsammans med agila produktutvecklingsfilosofier för att

testa om de tillämpar sig för ett affärsfall som kräver billig och enkel applikationsutveckling. Jag kommer utveckla en minimum-viable-product (MVP) mobilapplikation som skall kunna lanseras på Android och iOS plattformarna. Applikationen skall till början vara gratis och den skall följa jägarexamens uppsättning genom att ha olika kategorier med flervalsfrågor. Om applikationen visar sig värdefull för sina användare så kan den sedan göras betald för att hämta inkomster.

Genom detta utvecklingsprojekt ville jag hitta svar på följande forskningsfrågor:

- Hur väl ett cross-platform-ramverk, som React Native, tillämpar sig för att utveckla en mobilapplikation för flera plattformar.
- Hur mycket av koden som kan delas mellan de olika plattformarna.
- Hur väl existerande mjukvaruutvecklingsprocessteori stöder ett utvecklingsprojekt av en enkel MVP applikation som drivs av en ensam programmerare.

Mjukvaruutvecklingsprocessen fyra huvudaktiviteter är mjukvaruspecifikation, mjukvaruutveckling, mjukvaruvalidering och mjukvaruevolution. Mjukvaruspecifikation handlar om att kunder eller slutanvändare tillsammans med utvecklare definierar vad mjukvaran skall göra och vad den inte skall göra. Mjukvaruutveckling handlar om att utvecklare designar mjukvaran och sedan implementerar den. Mjukvaruvalidering handlar om att granska mjukvaran för fel och se till att den motsvara det som beställts. Mjukvaruevolution handlar om att underhålla och vidareutveckla mjukvaran för att garantera att den möter kundernas och marknadens förändrade behov. De ovannämnda huvudaktiviteterna kan vara mycket komplexa och de varierar i arbetsmängd, de innehåller oftast också flera underaktiviteter, som t.ex. projektplanering, arkitekturdesign, och enhetstestning.

Utvecklare utför oftast samma aktiviteter då de utvecklar mjukvara, men de utför dem inte i samma ordningsföljd. Mjukvaruutvecklingsmetoder beskriver i vilken ordning utvecklingsaktiviteter utförs. I mitt examensarbete delar jag upp mjukvaruutvecklingsmetoder i plandrivna metoder och inkrementella metoder.

Plandrivna metoder bygger på tanken om att man planerar varje utvecklingsaktivitet först och sedan utför dem en i taget, så att en avslutas före en annan påbörjas. Plandrivna metoder fungerar under antagandet att utvecklare vet eller kan komma åt all information de behöver då de planerar. Men, detta är sällan fallet i mjukvaruprojekt och man upptäcker

ofta ny information när man utför olika utvecklingsaktiviteter, vilket kan leda till att man måste gå tillbaka i processen och göra om tidigare aktiviteter.

Inkrementella utvecklingsmetoder bygger på tanken att man utvecklar mjukvara i versioner. Första versionen utvecklas snabbt och används som ett verktyg för att få feedback från kunden/slutanvändaren för vidareutveckling. Varje version går igenom specifikation, design, implementation och validerings aktiviteterna och resulterar i mer och mer funktionalitet tills mjukvaruprodukten tillfredsställer alla krav. Denna metod tillämpar sig bättre till situationer med flera okända eller snabbt förändrande variabler eller krav, vilket är orsaken till att inkrementella utvecklingsmetoder blivit så populära för utvecklingen av mjukvaruprodukter.

MVP tankesättet är väldigt likt inkrementella utvecklingsmetoder eftersom det grundar sig på idén om att snabbt bygga och lansera en första version av mjukvaruprodukten. En MVP skall bara innehålla den mest nödvändiga funktionaliteten för att kunderna skall kunna tänkas använda produkten. Första versionen av en MVP applikation används sedan för att testa en affärshypotes och lära sig om vad kunderna tycker om produkten. Ifall man har knappar resurser och inte har en utomstående kund, som genom sina krav styr produktutvecklingen, utan man utvecklar en egen idé. Då kan MVP tankesättet kan vara ett bra sätt att närma sig mjukvaruproduktutveckling. Det uppmuntrar en att skära ner på funktionalitet och hålla kraven måttliga så att man snabbt kan lansera en fungerande produkt och validera behovet för produkten, vilket passar extra bra för småföretag och ensamma entreprenörer.

Mjukvaruutvecklingsprocessen bygger på de fyra ovannämnda huvudaktiviteterna, som används så gott som av alla organisationer som utvecklar mjukvara. Men, om organisationen väljer att utnyttja en plandriven utvecklingsmetod eller en inkrementell utvecklingsmetod eller en blandning av båda, beror på situationen till hands, organisationens styrkor och svagheter, och produktens egenskaper.

Traditionella mobilapplikationer har utvecklats med verktyg som är dedikerade för en viss plattform. Om man utvecklar en applikation för iOS plattformen så måste man använda deras verktyg och samma gäller för Android. Att utveckla samma applikation på två olika sätt är tidskrävande och dyrt, speciellt för organisationer med knappa resurser.

Med cross-platform-ramverk kan man utveckla en mobilapplikation som kan översättas till flera plattformar. Att utveckla med cross-platform-ramverk är billigare och enklare men det kan hända att man inte kan utnyttja plattformens alla egenskaper, eftersom man utnyttjar plattformens teknologi genom ett extra teknologiskt lager. Detta kan hindra cross-platform applikationers prestanda och den så kallade ”look and feel” egenskapen, som hänvisar till hur applikationen känns när man använder den.

Det finns olika cross-platform-ramverk för mjukvaruutvecklare att välja mellan och de erbjuder alla ungefär samma fördelar. Vilket ramverk utvecklare väljer beror på applikationens egenskaper, kundens krav och mjukvaruutvecklarnas kunnande och preferenser. I detta projekt kunde jag själv välja eftersom jag inte hade någon kund. Därför beslöt jag mig för att använda React Native ramverket. React Native utnyttjar webutvecklingsbiblioteket React och programmeringsspråket JavaScript, som jag redan var bekant med från tidigare. Detta gjorde det lätt för mig att komma igång med utvecklingen av applikationen då ramverket kändes bekant från början.

Under utvecklingsprocessen tog jag stöd av mjukvaruutvecklingsprocessteorin som jag undersökt tidigare och av en React Native mjukvaruutvecklingskokbok. Jag var ett enmanslag som skulle programmera sin första cross-platform applikation med teknologi som inte var bekant från tidigare. Dessa omständigheter ledde mig till att utnyttja en inkrementell utvecklingsprocess där jag utnyttjade MVP tankesättet för beslutsfattande angående funktionalitet.

Min MVP inspirerade kravspecifikationen var simpel. Applikationen skulle bara ha den mest nödvändiga funktionalitet för att hjälpa blivande jägare förbereda sig för jägarexamen. Det behövdes ingen login eller databas, bara tre olika frågekategorier och en möjlighet att kolla vilka frågor man svarat fel på för att användaren skulle kunna lära sig.

Med dessa krav började jag designa och programmera direkt i skriven kod. Jag jobbade mig fram en komponent i taget tills jag hade implementerat all funktionalitet jag ansåg applikationen behöva för att den skulle vara till nytta för slutanvändaren. Det tog mig ungefär en månad att programmera applikationen färdig.
Efter det testade jag applikationen med hjälp av digitala verktyg på olika rutstorlekar för att garantera att applikationens layout fungerade dynamiskt. Efter att jag var nöjd med

testerna så beslöt jag att applikationen var färdig för lansering och att utvecklingsprocessen för första versionen kunde avslutas.

Som slutsats för mitt arbete kan jag konstatera att jag nådde målet med utvecklingsprojektet och lyckades svara på mina forskningsfrågor. Min mjukvaruutvecklingsprocess avslöjade att mjukvaruutvecklingsprocessteori inte är till stor nytta för ensamma utvecklare eftersom det är mer riktat åt team och större projekt. Däremot är mjukvaruutvecklingskokböcker till större nytta då de mera konkret demonstrerar, via exempel, teknologins egenskaper, vilket är mer relevant i små utvecklingsprojekt.

Utvecklingsprocessen avslöjade också att React Native ramverket är ett mycket praktiskt verktyg för utveckling av MVP-applikationer. Jag lyckades som. en oerfaren utvecklare använda ramverket för att snabbt och enkelt utveckla en applikation som fungerar på två plattformar. Till slut så delade applikationen nästan all kod mellan plattformarna, vilket bevisar hur effektivt och praktiskt utveckling med React Native är.

Att utveckling av mobilapplikationer blir mer effektivt är bra nyheter för små organisationer med knappa resurser. Det gör det möjligt för dem att utöva idéer som inte tidigare varit möjliga. Även nisch applikationsidéer kan anses vara möjliga då utvecklingskostnaderna inte är för höga.

Då jag genomförde detta projekt så sökte jag efter litteratur om MVP applikationsutveckling, men hittade ingen. Jag tror det finns rum för någonting mellan den utförliga mjukvaruutvecklingsprocessteorin som är skriven för team och de detaljerade mjukvaruutvecklingskokböckerna som beskriver teknologiska egenskaper via exempel. Det finns säkert efterfrågan för litteratur som tacklar problemen kring MVP applikationsutveckling.

Det har varit roligt att jobba på detta utvecklingsprojekt och försöka skapa en lösning till ett behov jag själv identifierat. Jag kommer fortsätta utveckla applikationen inkrementellt med stöd från MVP tankesättet för beslutsfattande.