Evaluating React Native and Progressive Web App development using ISO 25010

Ludwig Bäcklund, Oskar Hedén

Supervisor : Dennis Persson
Examiner : Jody Foo
Upphovsrätt


Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances. The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page [http://www.ep.liu.se/](http://www.ep.liu.se/).

© Ludwig Bäcklund, Oskar Hedén
Evaluating React Native and Progressive Web App development using ISO 25010

Ludwig Bäcklund
Linköpings universitet
ludli839@student.liu.se

Oskar Hedén
Linköpings universitet
oskhe359@student.liu.se

ABSTRACT
The vast supply of different smartphone makes and models, along with their accompanying operating systems, increase the demand for an all-in-one development solution. Quite a few approaches to solving this problem have cropped up over the years, ranging from purely web-oriented solutions to something more akin to a native application. React Native and Progressive Web App development are two different approaches, both new and promising, on this spectrum. This thesis evaluates these approaches in a standardized way using the ISO 25010 Product Quality Model to gain insight into these types of cross-platform development as well as how well such an evaluation works in this context. The results show that, while not a perfect fit, a standardized evaluation brings forward less obvious aspects of the development process and contributes with a helpful structure to the evaluation process.

Author Keywords
Cross-platform; Development; PW A; Progressive Web App; React Native; ISO 25010

INTRODUCTION
Since their introduction in the late 2000’s, smartphones have increasingly dominated the mobile phone market. This has resulted in a multitude of devices from a wide range of manufacturers. The need for developers to satisfy all the different makes and models on the market has resulted in divergent development tooling and languages, like Objective-C for iOS or Java for Android. In order for a company to reach a wide audience they are as such required to construct the same app several times, one for each major operating system. This results in more man-hours being spent and more niche knowledge required within the company. As a result, the interest in cross-platform development is increasing due to the potential benefits of reaching more platforms with a single code base.

In 2015 Facebook released a framework with the goal of unifying web and mobile development called React Native1. Since its inception it has gained great traction among developers from a wide range of backgrounds due to its relative simplicity and novel workflow. The concept behind it is to write app code in JavaScript, more specifically by utilizing the React2 framework for the UI and plain JavaScript for the business logic, and then running that code on iOS and Android. A caveat to this is that not all code is cross-platform compatible and as such some platform-specific code will have to be implemented to get the app to run.

Around the same time, the term Progressive Web App (PWA) was coined to refer to something quite different, a web application that had the capabilities of adapting to a mobile environment. To qualify as a PWA, an application needs to, among other things, be fully responsive on mobile devices, provide complete offline functionality and serve content securely. At the core of a PWA lies the Service Worker which acts as a middleman between the application and the outside world. Here it can modify network requests, cache network responses and handle push notifications.

Motivation
For this thesis our goal is to evaluate cross-platform development in the form of a PWA or a React Native app. By using product qualities found in an ISO standard we will evaluate our two different cutting edge cross-platform development techniques and discuss emerging issues in relation to the qualities.

Due to the young age of both React Native and PWAs, there is no abundance of previous research that extensively evaluates the capabilities and shortcomings of these cross-platform solutions. Earlier studies have aimed to evaluate React Native in a more general sense [1]–[3] by developing and benchmarking small, less complex apps. Our focus will instead lie on evaluating cutting edge cross-platform development in a more standardized way, and the considerations of using that approach.

The possibility of unifying web and mobile through the use of a cross-platform framework like React Native or through developing PWAs is interesting both in an academic sense and from a business perspective as the reduced need for niche knowledge and fewer developers can lead to a more efficient use of resources.

Aim
This study is a result of work carried out in conjunction with IT-Bolaget Per & Per3, an IT consulting firm developing in-house for clients in the forestry sector. They develop and maintain specialized native iOS and Android apps for customers in the Nordics that adapt to their various needs. A goal for Per & Per going forward is to investigate how their current and future applications can adapt to new platforms. This allows customers to commit fully to their app ecosystem on all relevant platforms, not just iOS and Android.

1https://facebook.github.io/react-native/
2https://reactjs.org/
3http://2xper.se/
We will be implementing a smaller, partial version of Per & Per’s main app where we duplicate certain features in a PWA and a React Native app. The features we will implement mainly revolve around an interactive map implemented with a popular map framework, Mapbox GL, that displays a significant amount of data.

Delimitations
The current full version of Per & Per’s main app contains functionality and data that is beyond the scope of this study and therefore we will not be implementing all of its functionality in our test apps. In turn, this lets us spend our available time more efficiently, ultimately resulting in a more focused thesis. This is not meant to be an exhaustive investigation, we can only evaluate that which we encounter naturally (or can extrapolate from our experience) during development.

THEORY
Before introducing our research questions, found at the end of this section, we provide necessary theoretical familiarization by introducing different approaches to cross-platform development, the technical frameworks used during development, a model introduced by ISO that categorizes product qualities, and related work examined during this thesis project.

Cross-platform development approaches
Since the interest for cross-platform development increased in tandem with the spike in use of different mobile devices, several methods have been introduced to solve the issue of reaching more platforms using a shared codebase.

Raj et al. [4] presents four classifications of cross-platform development approaches often referred to in other similar articles, each with different advantages and disadvantages. The four classifications presented are:

- Web Approach
- Hybrid Approach
- Interpreted Approach
- Cross Compiled Approach

The Web Approach consists of using a web application designed to execute in the browser of the mobile device. Using this approach, the device will not need to have any components specific to the application installed, as the application will be browser based and the data in the application is server driven.

The Hybrid Approach is designed to serve as a middle ground between the web application and the native application. The approach uses web technologies to develop the app which then is executed within a native container on the device. The approach uses an abstraction layer to expose native device capabilities as a JavaScript API. Hybrid applications need to be downloaded and installed on the device.

Using the Interpreted Approach, the application code is deployed on the device, and the interpreter executes the code at run-time. The interpreter interprets on run-time over different platforms, while the logic of the application is captured in an platform-independent way.

The Cross Compiled Approach uses a cross compiler which converts the application source code into native binaries. Developers write the source code using a common language, and the cross compiler is responsible to convert the source code into the particular source code suited for each device. The cross compiler’s reliability and efficiency is at the essence of the whole approach.

Technical Frameworks
Throughout the different cross-platform development approaches there is a myriad of different frameworks available, each introduced to solve a specific problem, often, but not necessarily, associated to a specific approach.

React Native
React Native stands out among its cross-platform competitors with its ability to render native components on both supported operating systems, iOS and Android, while still maintaining a unified code base. Most other cross-platform solutions either utilize a purely web-oriented or hybrid approach. In contrast, React Native uses something closer to an interpreted approach, where app code is run separately from native code through the use of an interpreter [4]. In the case of React Native, a JavaScript interpreter called JavaScriptCore is used. The process behind the rendering of a React Native app is similar to that of React. React uses what is called the virtual DOM (Document Object Model) which is a programming concept where an ideal representation of the application UI is kept in memory to be synced with the real DOM whenever something is added or updated. This process of syncing is referred to as reconciliation. Although React Native keeps a similar virtual representation in the background, it differs in how that representation is applied when rendering. A mobile app does not have a DOM in the same way a web application does, instead React Native directly calls the API’s of iOS and Android to render the appropriate components.

All communication between native code and JavaScript goes through the so-called bridge, more specifically the message queue. Messages from both sides get serialized and batched before being sent via the message queue to deliver information about network requests, creating views, layout updates and more. This communication is asynchronous which means all communication is in the form of method calls or callbacks. The message queue can become congested if too much information is being passed at once and this makes it so an app becomes unresponsive in terms of handling new actions while still being interactable by the user.

As previously mentioned, React Native uses an approach similar to the Interpreted Approach. React Native uses the JavaScriptCore as an interpreter between the JavaScript controllers written in React Native, and the native counterparts. This means that the native UI components are working on separate threads from the React Native code. One of the great

---

4http://trac.webkit.org/wiki/JavaScriptCore
5https://reactjs.org/docs/faq-internals.html
advantages of React Native is that since the bridge is asynchronous between different threads the JavaScript code won’t cause the UI to freeze or slow down.

**Progressive Web App**
The term Progressive Web App was initially coined by a Google software engineer in 2015 to describe a new kind of web application. At its core, a PWA is essentially a web application that fulfills a certain set of requirements specified in the [PWA checklist](https://developers.google.com/web/progressive-web-apps/checklist) maintained by Google. These requirements exist to ensure that a PWA runs fast and functions reliably, ultimately resulting in something that, to a degree, resembles a native mobile app. In addition, when these requirements are met for an application, its users can be prompted to add a shortcut to the application on the home screen of their mobile device. Using this shortcut brings up the application as if it were a native application, without usual browser elements like the address bar.

PWAs use the Web Approach as the app essentially is a web application accessed via the device’s browser. What makes PWA different is the use of service workers which allows the app to utilize more functionality than what is traditionally available through browsers.

**Service Workers**
The concept of a Progressive Web App is tightly bound to what is known as a service worker. A service worker is a script that runs in the background of a web application at the browser level to enable additional functionality like push notifications, application caching and background sync. Service workers are asynchronous by nature to not keep the main JavaScript thread busy, and are terminated and restarted periodically depending on how they are being used. Due to their asynchronous nature, service workers have no access to the DOM and instead all communication between service workers and their controlled pages goes through a message passing system.

Service workers have a lifecycle that is entirely separate from that of the application. The first step of the lifecycle is the installing phase which is called once per service worker to signal whether the service worker managed to install or not. The service worker then goes into either the waiting or the activation phase, depending on if there’s already a service worker active for the application. Only one service worker can run at a time with currently running service workers are compared to new versions at application start-up to determine if an update is in order.

During normal operation of the application the service worker lies dormant in the background, waiting for relevant events to be fired. For example, if the service worker has been configured to act as an intermediary for network requests then it will intercept all outgoing requests and perform whatever operation is specified in the event handler. The nature of service worker functionality means that they operate on a low enough level to be useful for a broad range of application scenarios, rather than being heavily specialized for a few.

**Software quality evaluation**
Evaluating the quality of software is an important and much studied area of the development process. ISO 25010 [5] is a standard that partly sets out to categorize characteristics and sub-characteristics of applications into a Product Quality Model (PQM), seen in Figure 1, that lays the foundation for a robust software evaluation. By determining which of these characteristics apply to the software at hand and then evaluating them, one can come to a better understanding of the capabilities and shortcomings of any given software product.

The PQM consists of eight overarching characteristics, each made up of at least two related sub-characteristics. The following list outlines the definitions of the main characteristics, presents each related sub-characteristic, and provides a descriptive summary of the quality as a whole.

**Functional Suitability**
"degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions" [5]

Sub-characteristics: functional completeness, functional correctness, functional appropriateness

Functional Suitability depicts how the given set of functions covers all tasks and user objectives, to which degree of precision software provides correct results, and how well the system allows the user to complete tasks and objectives.

**Performance Efficiency**
"performance relative to the amount of resources used under stated conditions" [5]

Sub-characteristics: time behaviour, resource utilization, capacity

Performance Efficiency measures how well software handles time behaviour, like response times and throughput, and how different types and amounts of resources are utilized to meet requirements.

**Compatibility**
"degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment" [5]

Sub-characteristics: co-existence, interoperability

Compatibility describes how software can perform efficiently while sharing resources and environment with other software without having negative impact on either part and how software can exchange and use information with other software.

---

Usability
"degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [5]
Sub-characteristics: appropriateness, recognizability, learnability, operability, user error protection, user interface aesthetics, accessibility

Usability portrays how users can determine software’s ability to satisfy their needs, how easy software can be learned to be used, to which degree software can be maneuvered and operated and how well software prevents users from making errors. It also brings up how the user interface of software allows satisfying interaction in terms of aesthetics and how suitable software is for a wide range of users with different capabilities.

Reliability
"degree to which a system, product or component performs specified functions under specified conditions for a specified period of time" [5]
Sub-characteristics: maturity, availability, fault tolerance, recoverability

Reliability outlines how dependable software is during typical use and how easily it can be accessed when needed, along with how it functions when encountering errors and how it recovers from them.

Security
"degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization" [5]
Sub-characteristics: confidentiality, integrity, non-repudiation, accountability, authenticity

Security is about ensuring that only authorized persons can access and modify software, as well as determining exactly who was behind these actions and in which manner they happened.

Maintainability
"degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers" [5]
Sub-characteristics: modularity, reusability, analysability, modifiability, testability

Maintainability talks about how software can be modular to enable painless changes to individual parts, re usable to be easily placed in different systems, analyzable to find defects and estimate the impact of new functionality, modifiable to enable changes without introducing bugs and testable to easily verify that criteria are met.

Portability
"degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another" [5]
Sub-characteristics: adaptability, installability, replaceability

Portability is all about adapting software to its relevant environments and use cases, which includes how one can install and uninstall it as well as how cumbersome it is to replace the software altogether with a functionally equivalent alternative.

Related Work
Given the sparse research on the topic of modern cross-platform development tooling, we have found no general consensus regarding what could be classified as key areas of interest. However, by closely examining existing research on general cross-platform development we ourselves have formulated categorizations for such areas of interest, capturing reasons as to why cross-platform development can be a worthwhile pursuit, these being: (1) Maintainability, (2) Reduced resource expenditure, (3) Reduced need for niche knowledge.

Maintainability
Reasons concerning maintainability in the literature is mentioned in a general meaning, e.g. not mentioned in association with the ISO standard presented in the previous section. In other words, this key area does not aim to map to the characteristic maintainability found in the PQM.
One of the most attractive qualities of cross-platform development is the allure of its apparent maintainability compared to that of a purely native app. Especially highlighted reasons in the research is the reduced effort needed to maintain [2], [4], [6], [7] cross-platform apps due to their shared code base. Moreover, keeping a shared code base facilitates easier testing [6] and deployment of new features [2].

One aspect of this maintainability in the context of React Native lies in its component system where UI code is written once and then used seamlessly across platforms. A majority of code written for a React Native app has been found to be reusable, with Hansson and Vidhall coming to the conclusion that 75% of their code base was not platform specific [1]. Maria Nunes Andrade Lobo dos Santos arrived at an even greater degree of code reusability with their particular use case, over 99% [8].

The degree to which one can modify and extend code after it has been implemented is also a substantial motivational factor behind the decision to pursue cross-platform development. Abrahamsson and Berntsen tried to quantify modifiability in their study, referencing the PQM presented by ISO 25010, coming to the conclusion that React Native allowed for easier future modifications. However, according to the authors, the smaller sample size made it difficult to draw any major conclusions [6].

Resource expenditure
One of the most widely mentioned aspects brought to light when looking at why cross-platform development is being explored is that less resources would be required to develop and maintain feature equivalent apps for different platforms. Resources is in this case referring to the number of working hours [1], [3], [6], [7], [9], [10] and number of developers [11] required. Another aspect specifically mentioned is the time it takes for the app to reach the market [6], [9], which is preferably kept as low as possible.

Niche knowledge
Another aspect to cross-platform development is the idea that one would need less niche knowledge inside the development team to create an app that is required for multiple native platforms [1]-[3], [11], [12]. By only working with a single shared code base several papers point to increased productivity within the development team [8], [12].

Research Objective
The research questions of this thesis will firstly focus on using the Product Quality Model described in Software quality evaluation to evaluate cross-platform development and secondly on how well that evaluation works.

1. How well does PWA and React Native cross-platform development work in regards to the qualities presented in the Product Quality Model?

2. Is it reasonable to use the Product Quality Model to evaluate products of cross-platform development?

METHOD

This section explains the methodology employed in the development process of both apps, as well as how the PQM introduced in Software quality evaluation is adapted for use with these cross-platform frameworks.

Development
In order to identify the issues that arise when developing cross-platform apps in the form of a PWA or a React Native app, we developed one app for each of the two approaches, both featuring critical functionality desired by the stakeholder9 Per & Per. Initially we discussed what intentions, needs and goals Per & Per had for the project, i.e the stakeholder intentions [5]. After this initial meeting we moved on to establish clearer and more defined requirements, the stakeholder requirements, to use as system requirements (as per ISO 29148 [13]). Once the system requirements were specified we went on to implement each requirement in both of the apps according to the predetermined hierarchically ordered list of requirements, outlined within the stakeholder requirements.

To achieve the most conformity between the apps and the development experience, both apps were developed using React in some way, implicitly for the React Native app and added by us in the PWA. create-react-app and create-react-native-app were used to bootstrap the configuration process, and Mapbox GL for React and React Native10 to provide the map functionality. A pre-compiled style-object was used to style the map correctly, and custom authenticated tile-data requests were needed to receive the relevant map-data from dedicated servers.

React Native
Using the Mapbox GL package introduced for React Native, we managed to implement an initial map using Mapbox GL's own styling. Moving forward we implemented state management, which in turn would allow us to dynamically manage map data using the Redux11 state. By using mock data, i.e. data produced by us during development with the purpose of mimicking actual data, we managed to display desired geometric representations of map data as expected still using default styles provided by Mapbox. We also implemented the ability to draw new geometries in the app in real time.

Moving onto dealing with real data, we encountered some issues. Unlike the JavaScript version of the framework, Mapbox GL for React Native included no method in their API to modify the tile-requests in the way we required in order to authenticate to the servers. A possible solution to this would have been to modify the native SDK:s that the React Native Mapbox library depends on, but we chose not to pursue this approach.

Progressive Web App

9stakeholder - individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations [13]
10https://github.com/alex3165/react-mapbox-gl
https://github.com/mapbox/react-native-mapbox-gl
11https://github.com/reactjs/redux
For the development of our PWA, we used the Mapbox GL library made for React, and, similarly to React Native, we implemented our state management using Redux. The initial stages of the development were made using standard Mapbox styling and mock data. Like React Native Mapbox library, the Mapbox library for React contained no API method which enabled a transformation of the fetching request. However, using the service worker’s ability to act as a proxy between the app and all network traffic, we could modify the fetch requests going out from Mapbox using standard web techniques. We managed to retrieve actual map data from the client servers alongside the data that represented different properties of users and portrayed them in the app using the pre-compiled style guide.

Once we tried the app on different platforms we achieved desired behavior for the most part. Service workers are relatively new and some browsers do not support all of its functionality properly as of May 2018, e.g. no push-notifications of Safari. Another aspect with developing a web app to use on mobile devices is the mapping between browser events. The subtle difference between a click in a web browser, and a press in a mobile device browser translates into different browser events firing.

Adapting the PQM

In previous research, evaluating a cross-development approach or the resulting products has been mostly a subjective process with ad-hoc evaluation methods. Although Furuskog and Wemyss [12] justifies their choices of performance metrics by looking at previous research this is not always the case. Danielsson [3] evaluates their app by looking at how well the app could be replicated in React Native, alongside looking at performance in terms of personally chosen metrics like GPU frequency, CPU load, memory usage and battery consumption. Similarly, Hansson and Vidhall [1] evaluates uses CPU usage, memory usage, frames per second, response time and application size. Neither paper discusses methods previously employed in previous research. These divergent methods and their corresponding results makes it hard to compare the conclusions made in the papers against one another.

To try and bring some kind of structure to this evaluation process we use the PQM. The traditional use case of the PQM is to use it when evaluating against a finished product to see how it functions in relation to the listed characteristics and sub-characteristics. We adapted this process to instead focus on cross-platform frameworks and their resulting product, in this case as a PWA or a React Native app.

We adapted the eight main characteristics from the PQM, presented in Theory, to shift the focus to evaluating the PQM from the perspective of cross-platform frameworks. We interpreted each characteristic summary in Theory to better fit with cross-platform frameworks instead of products. The following sections contain all of these interpretations.

**Functional Suitability**
The functional suitability of a cross-platform framework lies in its ability to provide important functions for cross-platform development like exposing relevant hardware API’s to enable additional functionality. It also extends to which degree the presented functionality generates correct results in regards to the developer’s goals.

**Performance Efficiency**
The performance efficiency of a cross-platform framework depends upon things like how the implementation behind the framework affects performance aspects of the developed apps, e.g. the threading capabilities of the programming language being used for development. It also includes the resources required to run an app built with the framework, for example how much memory is utilized when running.

**Compatibility**
The compatibility of a cross-platform framework concerns its ability to efficiently share resources and environments with other frameworks without impairing either party, and how well it can exchange and use information with other frameworks.

**Usability**
The usability of a cross-platform framework depicts how well a developer can identify the plausibility of using that framework to solve a problem, and how much effort it takes to learn the ins and outs of a framework for developers, which oftentimes is connected to the thoroughness of the documentation. Moreover, it involves how explicitly stated the less obvious properties of the framework are, e.g. if the documentation highlights common pitfalls. Another aspect of usability can be how the user interface, in this case developer interface (API) of the framework, allows for satisfying interaction, and if the framework is adapted to suit developers of a wide range of capabilities.

**Reliability**
The reliability of a cross-platform framework is determined by how mature it is, i.e. how well it performs during typical use, as well as how well the framework can be accessed when needed and how the framework handles errors and shortcomings and what tools are available to recover from them.

**Security**
The security of a cross-platform framework deals with who is authorized to access and modify the underlying code of the framework, i.e if it is open-source or not, as well as the monitoring of who makes changes and how they happen, i.e. if the framework use any type of version control to monitor the development.

**Maintainability**
The maintainability of a cross-platform framework describes to what degree it can be used to seamlessly change individual parts, and how well it can produce reusable components to be placed in other systems. Furthermore, it looks at how easily the framework can be analyzed to find defects and assess the repercussions of new functionality, as well as how it allows for easy modification, without adding bugs in the process, and verifiable testing based on given criteria.

**Portability**
The portability of a cross-platform framework encapsulates its ability to be used to implement apps on different hardware and platforms, how easily installable it is in a developer environment and how one could replace a framework with another while not losing any functionality.

RESULTS
This section will cover our issues and experiences with cross-platform development using React Native and Progressive Web Apps in relation to relevant qualities presented in the PQM.

Although the term PWA does not refer to a framework in the same way that React Native does, we will use the terms "cross-platform framework" and "framework" as umbrella terms to encapsulate both React Native and PWA for simplicity’s sake.

Functional Suitability
One of the first aspects one might look at when deciding which cross-platform framework to chose can be related to how suitable the framework’s functionality is to solve the given problem. By looking at the framework’s API the developer can get initial insight into what the framework has to offer.

While React Native exposes certain native functionality through its API, complete functionality cannot be guaranteed. It is however possible to utilize platform specific functionality through the implementation of native modules. This allows you write native code for iOS or Android which then can be called via JavaScript code. Of course, this undermines the goals of shared code and reduced niche knowledge as you will have to know how to develop for both platforms to do this.

PWA functional suitability is hurt by its lack of hardware access. While it can still run on any hardware, it cannot take advantage of its underlying capabilities to provide additional functionality for the end user. For us this meant that some of the desired functionality could not be implemented as it existed in the original app due to the limited hardware access. Furthermore, the fact that browser events generated on desktop devices are different from those generated on mobile devices can lead to problems. One example of this from our process was how touch browser events were more limited than the available mouse equivalents, making it hard to handle advanced interactions by the user.

Performance efficiency
The performance that a cross-platform framework affords can vary between inconsequential to absolutely crucial depending on the application that is being developed. Some apps, for example a news reader, do not particularly care how much performance can be squeezed out of a device while an interactive map app does.

React Native has the advantage of being able to utilize native components by calling their APIs to handle rendering, instead of working with layers of abstraction that can reduce performance. The perceived performance of React Native is also special because of just that, if the JavaScript thread becomes congested the app is still fully interactable and smooth to the touch as the main thread is not affected. If worse comes to worst then there is still the option of implementing a native module to directly access the capabilities of the platform in question, bypassing the troublesome parts. All of this together means that the expected performance of our React Native app, more specifically the interactive map in it, is close to that of native performance as the Mapbox library in React Native is built upon the native iOS and Android Mapbox SDK’s.

The nature of our PWA being a web app means that it is abstracted away from the device it runs on and as such it cannot take advantage of device optimizations to increase performance. Being a web app also brings with it the need for a browser to be running the app. A browser will always have a certain overhead to it as its functions, not necessarily related or useful to the app, still needs to run in the background. Our PWA is also fully powered by JavaScript, a single-threaded language, which inherently limits its throughput. This means that the expected performance of our PWA is potentially significantly lower than that of our React Native app due to missing out on crucial low-level optimizations that help the performance of our interactive map.

Compatibility
The ability to co-exist with other cross-platform frameworks to exchange and use information between frameworks without compromising either framework can prove vital, e.g. in cases where a single framework is not able to suit all the needs of the task at hand.

During our thesis work, however, the need for compatibility was non-existent in the scope of the task that we set up. Thus, this characteristic did not emerge naturally during the development process, given the parameters of our particular use case.

Usability
The ability to quickly learn and determine the feasibility of using a new framework can weigh heavily when looking at possible approaches to solving a problem.

We both had previous experience of working with React and React Native which helped during development, but even if we had not had that experience, the active community of React developers simplifies the learning process. In fact, this is one of the strengths of the framework.

Implementing our PWA did not require us to learn or adapt to any new paradigms, programming languages or coding conventions. The app could be developed using any web framework preferable to us as developers, which in our case resulted in using React. Therefore, the learnability of using our PWA was quite high as the only requirement of using PWA is that we abide by certain parameters, all provided in the previously mentioned PWA-checklist maintained by Google.

A common issue with both approaches when it comes to usability is the limited amount of documentation available. Since both React Native and PWA development are relatively
new techniques, the amount of documentation on respective approach is lacking, especially when it comes to more niche use cases. Although there are oftentimes alternatives on the market, they rarely contain anything more than basic examples in their documentation, and if your situation requires a special aspect of their solution to work differently it might be hard to find accurate help in the documentation or on other forums.

Reliability
Whatever reasons one might have to use any specific development approach, the ability to trust that the tools contained within that particular approach can reliably deliver needed functionality is paramount. For both of our chosen approaches, the inclusion of open-source libraries in the development process is common, and even encouraged at times. Since open-source libraries are subject to their maintainers' level of motivation, continued development and maintenance cannot always be ensured.

React Native was released as open source at the time it hit the markets in 2015. With more than 1600 contributors as of May 2018, several of which are based at Facebook, React Native’s support is continuously growing and as such the library gets more reliable as time goes on and functionality is tested. Therefore, the otherwise frightening idea that developers might abandon development for the library is not as bad. However, many of the other libraries aimed at solving issues for React Native are not as well backed, which means that a product built with React Native might end up with components that are no longer maintained or even deprecated.

PWAs use service workers which are backed by a W3C standard. This gives more weight to the reliability of using a PWA. On the other hand, browser developers are not enforced to implement the functionality service workers provide. This means that the app might act differently depending on the browser, and that in some cases most of the functionality in the app will be unsupported due to a lack of service worker implementation in the browser. As an example, during our development Apple released their first patch to Safari that supported service workers. Even still, the support in Safari is not complete as functionality such as push-notifications and background sync is not supported in that release.

Security
Apps often deal with sensitive information and this makes it important to be able to ensure that every part of the development pipeline meets security standards, including the framework that powers it.

React Native is an open-source framework with a large amount of contributors and while some of them are employed by Facebook, others are just developers who feel like they can contribute to the project. While most have good intentions, there might be some with sinister goals in mind and this could compromise the integrity of the framework if not handled properly. Luckily, as with most larger projects, new code is vetted by trusted contributors and checked for anything out of place which drastically reduces the risk of malicious code being folded into the framework.

The core of our PWA is the service worker and with its implementation being managed by the companies behind Chrome, Firefox, Safari and other browsers, one can be reasonably sure that it is handled proficiently. There is of course the problem of proprietary software not being open to inspection if a closer look at the internals is desired so there is a trade-off to be made there.

Maintainability
One of the strongest arguments for using cross-platform development lies in its ability to be more easily maintained over several platforms than having several native code bases for the same app.

React Native is very modular in its component system in that components easily can be altered without affecting other components in the app. As the components can be made in a very decoupled fashion, mostly by using props to inject data into components, the ability to seamlessly alter parts of the app increases dramatically. React Native is reusable in that components easily can be used again within other React components using a simple include. In a similar way, related React components can easily replace each other.

Depending on the implementation, PWAs can be implemented in a similar fashion using different techniques. The freedom of choosing the technique preferable to the developer determines the possibilities of building modular and reusable components.

Furthermore, being able to analyze and debug software during development is critical to save time and to reduce the risk of leaving behind bugs in the code. It also plays an important role in the introduction of new developers and maintainers to a project since without the proper tooling, it is hard to get a grasp of how a program operates behind the scenes.

With our PWA essentially being a normal web application with extra functionality added on, there is only a limited amount of ways to debug it. Mobile device browsers do not have any extensive development tools available. You can choose to log something to a console that you debug remotely which, while easy, is limited in how much information it can present in a readable manner at once. The other option is to make use of traditional web application debugging techniques like adding breakpoints in the code. These are then triggered and pauses the application whenever it gets to that point during execution, which lets you inspect variables and other data.

Similarly, for our React Native app we can debug the JavaScript code via the same methods as the PWA, along with other helpful tools such as a built-in inspector to analyze all rendered components. But as React Native produces actual native builds of the app, we can use iOS and Android tools like Instruments or Traceview to investigate the native parts of the app. This allows for a more in-depth look at our app and its resource utilization.

https://www.w3.org/TR/service-workers-1/
Portability
Applications have varying needs when it comes to platform portability. If widespread public use is sought after then, naturally, reaching out to as many platforms as possible is optimal. Some applications are instead more focused with niche workloads or small customer bases where platform capabilities outweigh the need for widespread use. Where your app fits in on this spectrum, along with its technical requirements, determines what cross-platform tooling is appropriate.

React Native runs exclusively on Android and iOS devices as a result of how it works behind the scenes. Our React Native app is in reality an iOS and Android app that interfaces with the written JavaScript code. React Native exposes the native API’s of iOS and Android to render components, while keeping the application state on the JavaScript side.

Our PWA can run on all platforms that field a browser with service worker capabilities. As of May 2018, an estimated 80% of browser users have access to service workers. With our apps being proof-of-concepts we had no strict need of supporting any particular set of platforms but the context of our task implicitly relates to the platform availability of the investigated solutions.

The portability of both cross-platform solutions depend on something out of reach for a lone developer. In the case of React Native it is dependant on what platforms the core team determines to be worthwhile to develop for, possibly influenced by the market share of mobile operating systems. However, with React Native being open-source it is technically possible to implement extra platform support independently. Progressive Web Apps depend upon service workers and their browser implementations which are standardized and thus cannot be externally implemented.

DISCUSSION

Method
Development
The development process was tightly bound to the task we were assigned by Per & Per, and as we both had experience in working with React and React Native we started off development with React Native. The development then transitioned from the React Native app to the PWA when we came to the point of having to implement patches to the native Mapbox SDK’s. These patches had already been implemented by Per & Per for use in the native applications they maintain and as such it was deemed not useful, either for the thesis or the task at hand, to do the same for our app.

As our project shifted when we found problems with the first approach we broadened our scope to include the investigation of PWA in addition to React Native. As we did, our evaluation method changed from focusing on measuring app performance to instead investigate development frameworks. In relation to React Native, none of us had any experience in working with PWA.

Adapting the PQM
Due to time related and budgetary limitations, we were unable to access the full SQuaRE series of ISO standards, which ISO 25010 is a part of, to use during the evaluation process. The other standards in the series might have helped solidify our method more, depending on if they were applicable to our evaluation method with the PQM.

With the characteristics in the PQM being so broadly defined there is a question of how to interpret them when being adapted to the scenario at hand. With our scenario being cross-platform frameworks instead of a finished product it meant that the adaption of the qualities could be rather liberal at times, as some were not a perfect fit when evaluating the way we did. Particularly functional suitability and compatibility were hard to adapt to our method and are generally more suited to evaluating finished products. With that in mind, one danger of adapting the PQM could be that we might introduce subjective bias to the adapted definitions. We tried to mitigate this effect by altering the original definitions as little as possible, but the human factor remains.

Results
Our results are directly bound to the scope of our task and as such we did not manage to touch upon all the qualities in the PQM during the development process. The functionality available in the apps is not substantial enough to guarantee that all qualities are evaluated to their fullest potential.

As the evaluation in our thesis changed during the project we had a different focus during the early stages of the development. This could mean that we missed some aspects of our evaluation in relation to the PQM during this time. Seeing as how the study was of the exploratory nature, it could have been better to devise our primary method accompanied by a secondary method before implementation began. Thus, aspects of the secondary method could easier be kept in mind even through using the primary method.

CONCLUSION

The purpose of this thesis was to evaluate cross-platform development in a standardized way to firstly find out how the two cross-platform frameworks fared in regards to the PQM qualities, but also to find out if a standardized evaluation like this was a viable approach.

In summary of our results, React Native outperformed PWA in regards to Functional Suitability with the possibility of implementing native modules while PWAs hardware access is lacking. The abstractions needed for running PWAs translates into a worse Performance Efficiency compared to React Native’s ability to call native APIs. Both React Native and PWA showed a high Usability in terms of the framework itself, although the documentation available to both frameworks is rather limited due to their age. Since PWAs can utilize the full potential of tried and tested JavaScript ecosystem its Reliability is better than that of React Native which requires brand new solutions. However, third party package reliability can vary greatly which influences the reliability of a finished product. In terms of Security both frameworks are rather open to the public eye with the service worker standard

---

13https://caniuse.com/#feat=serviceworkers
for PWAs and the open-source development of React Native but with the complete code being available only for React Native, it is more open to scrutiny. **Maintainability** is one of the most promising characteristics for both frameworks. React Native is very modular in its component system, and PWAs allow for similar flexibility in its solutions given the preferences of the developer. Finally, with PWAs being able to run on many different platforms, while React Native only runs on two, it also has a higher **Portability**.


Using the PQM presented in ISO 25010 for the purposes of evaluating a cross-platform framework is not the typical usage scenario for it. The characteristics presented are rather intended for the purpose of evaluating a finished product. With that said, the evaluation of React Native and PWA development in the context of this standardized quality model has uncovered more about the less explored parts of both frameworks than we would be able to find with a less stringent approach. While some characteristics of this particular quality model were hard to adapt to the evaluation of cross-platform frameworks, we still think that the value that standardized evaluation brings to the process is reasonable and worth exploring further. The structure the PQM provides helps reduce the subjective nature of an evaluation and makes it easier to compare and summarize results from different authors.

**Future Work**

During our thesis we were unable to investigate our evaluation method as much as we would have liked, chiefly due to time restraints. As a result, our first recommendation for future work is to look at cross-platform development using the entire **SquaRE**, i.e. the ISO 250XX, series of standards as grounds for the evaluation process. We believe that this could help solidify findings further and expand the scope of the evaluation.

Something else that would be interesting to investigate is how one could design an application that explicitly sets out to test all of the characteristics in the quality model, for a balanced evaluation. A possible downside to this is that the results could be interpreted as more artificial when the application is formed after the evaluation method rather than a real-world scenario.

**References**


