Evaluation of Cross-Platform Mobile Development Tools

Development of an Evaluation Framework

Linus Öberg
Abstract

The aim of this thesis is to determine what cross-platform mobile development tool that is best suited for Vitec and their mobile application "Teknisk Förvaltning". But more importantly we will in this thesis develop a generic evaluation framework for assessing cross-platform mobile development tools. With the purpose of making it easy to select the most appropriate tool for a specific mobile application.

This was achieved by first, in consideration with Vitec, selecting Cordova + Ionic and Xamarin.Forms as the cross-platform tools to be evaluated. Then secondly by proposing an evaluation framework containing criteria evolved from discussions with mobile application developers, experience gained by developing prototype applications using the tools to be evaluated and through literature studies. The initial framework was also refined by investigating what other research exists in the area and by learning from their methods and conclusions we refined our initial evaluation framework.

The result of performing the evaluation using the suggested evaluation framework is visualized in a result table, where the fulfillment of each criteria is graded from one to five. By allowing Vitec to rank how important these criteria are for their application, we were able to determine which of the tools that was the best fit for their application. We have thereby succeed in developing a generic evaluation framework which can be used to assess any cross-platform mobile development tool.
## Contents

1 Introduction 1

1.1 Vitec 1

1.2 Outline 2

2 Background 3

2.1 Mobile operating systems of today 3

2.1.1 Android 4

2.1.2 iOS 5

2.1.3 Windows Phone 6

2.2 Cross-platform development 7

2.2.1 Cross-compiler approach 7

2.2.2 Web approach 8

2.2.3 Hybrid approach 9

2.3 Cross-platform development tools 10

3 Problem Description 11

3.1 Problem statement 11

3.2 Cross-platform tools and frameworks we will evaluate 12

3.2.1 Xamarin 12

3.2.2 Apache Cordova/PhoneGap 14

3.2.3 Ionic 14

3.3 Methods 15

4 Evaluation Framework Proposal 17

4.1 Development perspective 17

4.2 User perspective 18

5 Evaluation Framework Enhancement 19

5.1 Related evaluation frameworks 19
5.2 Enhancement of our evaluation framework
5.3 Refined evaluation framework

6 Evaluation
6.1 Prototype applications
6.2 Part 1 - Properties
6.3 Part 2 - Development perspective
6.4 Part 2 - User perspective

7 Evaluation Results
7.1 Best cross platform development tool for Vitec

8 Conclusions
8.1 Future Work

Acknowledgments

References
1 Introduction

Mobile devices like smartphones and tablets are today ubiquitous. According to Gartner smartphone sales to end users totalled in 1,244,890,000 units under 2014, that is 28.4 percent more than 2013 when 969,721,000 units were sold. This increase of mobile devices has resulted in that over the past several years digital media consumption on mobile devices like smartphones and tablets have skyrocketed. According to Kate Dreyer at comScore the amount of digital media time spent on a smartphone has increased with an impressive 394 percent from 2010 to 2014.

Mobile devices have become something we use every day to access various digital services anytime, anywhere and anyhow. Completely natural. Whether it comes to listening to Spotify, sending selfies, pay the bills, or meet a potential partner through the dating application Tinder, we today use our smartphone.

But to build native applications to each mobile platform can be very expensive, especially if the application has relatively few users. This thesis will therefore evaluate a couple of cross platform mobile development tools, but more importantly we will develop a generic evaluation framework used to evaluating these different tools. The evaluation framework will be able to be used to evaluate any cross platform development tool, with the aim to make it easier for developers to select the most appropriate tool for a specific mobile application. This will be accomplished through extensive literature studies, implementations and evaluations of prototype applications and interviews with developers.

1.1 Vitec

This thesis was done in collaboration with Vitec. Vitec is a Nordic software company founded and headquartered in Umeå since 1985. They are specialists in standardized software for industry-specific needs and their businesses areas are: real estate, energy, health, estate agents, finance and insurance, car parts and media.

To one of their business applications, Teknisk Förvaltning, there also exists a mobile application that needs to be available for customers on both Android, iOS and Windows Phone. But since their mobile application has relatively few users Vitec has, for both time and cost reasons, chosen to develop the application using Apache Cordova, which is a framework for building hybrid mobile applications using web technologies, such as HTML5, CSS3 and JavaScript. This means, in short terms, that the same application can run on all needed platforms using one single codebase.

http://www.vitecsoftware.com/
This was exactly what they needed, a way to develop their mobile application to multiple platforms at the same time, in order to save both time and money. But their current application has deficiencies in terms of both functionality and performance. Therefore, they are now interested to find out what other solutions there exists and which one of these solutions that is best suited to fit their specific needs.

1.2 Outline

This is an overview of the chapters in this master thesis.

1. **Introduction**  
   Describes in short the motivation and the purpose of this thesis. Along with a quick glance at the software company Vitec who this thesis was done in collaboration with.

2. **Background**  
   We look at what the major mobile operating systems of today are and what approaches and tools there exists when it comes to cross-platform mobile development.

3. **Problem Description**  
   We define the problem statement and state what cross-platforms mobile development tools we will evaluate in this thesis. We will also present what methods that will be used to answer the problem statement.

4. **Evaluation Framework Proposal**  
   We purpose an evaluation framework for evaluating cross-platform mobile development tools. This initial evaluation framework have been developed thought discussions with developers at Vitec along with experience gained by developing prototypes using the different tools to be evaluated and by comprehensive literature studies.

5. **Evaluation Framework Enhancement**  
   The initial evaluation framework will be refined by looking at what other research there have been done on this area and by learning form their methods and conclusions we will refine our initial framework. The final evaluation framework will be presented in this chapter.

6. **Evaluation**  
   In this chapter the evaluation of the selected cross-platform development tools will be preformed by using our developed evaluation framework from the previous chapter.

7. **Evaluation Result**  
   The result of the evaluation will be visualized in this chapter. Also, by ranking what criteria Vitec consider important for their application and by applying this to our evaluation result, we will be able to determine which tool that is the best fit for Vitec and their application.

8. **Conclusions**  
   We will summarize and draw some conclusions from this master thesis work and discuss what future work there exists.
2 Background

The first handheld mobile phone prototype was invented by Martin Cooper, an engineer at Motorola in 1973. It weighed 1.1 kg and measured 23 cm long, 13 cm deep and 4.45 cm wide. It had a talk time of 30 minutes and did not run any applications [11]. The first phone that can be referred to as a smartphone was the IBM Simon. IBM Simon was developed and released by IBM in August 1994. The phone had a touchscreen, was able to send and receive faxes and emails, and had among other things both a calculator, notepad, and a calendar [23]. But the first phone that resembles today’s smartphones came first in 2007 when Apple Inc. introduced the iPhone [48]. One year later the first Android phone was released, it was manufactured by HTC and was called HTC Dream [47]. Much has happened since then and today smartphones exist everywhere and in a highly fragmented landscape where different devices are running different operating systems and have different hardware functionality in terms of screen size, sensors, input methods, etc..

2.1 Mobile operating systems of today

Not all smartphones run the same operating systems (OS). There are currently three major OS for mobile devices, Apple Inc’s iOS, Google’s Android, and Windows Phone from Microsoft. Android is the largest, with an almost 80% market share, iOS has about 20% and Windows Phone around 3%. All these OS support different Integrated Development Environments (IDE) and utilize different programming languages and Software Development Kits (SDK) for developing native applications to their OS.

Table 1: Comparison between the three major mobile operating systems of today.

<table>
<thead>
<tr>
<th>Company</th>
<th>Android</th>
<th>iOS</th>
<th>Windows Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launched</td>
<td>OHA/Google</td>
<td>Apple Inc</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Current Version</td>
<td>6.0</td>
<td>9.0.1</td>
<td>Windows Phone 8.1</td>
</tr>
<tr>
<td>Market Share</td>
<td>78.0%</td>
<td>18.3%</td>
<td>2.7%</td>
</tr>
<tr>
<td>Language</td>
<td>Java (Android)</td>
<td>Objective-C/Swift</td>
<td>C#/C++/VB</td>
</tr>
<tr>
<td>IDE</td>
<td>Android Studio</td>
<td>X-Code</td>
<td>Visual Studio</td>
</tr>
</tbody>
</table>

This means in order to build an application that supports all three major platforms the same application have to be developed three times. But fortunately, there are ways that make it possible to develop the application once and still support all major platforms, this process is called cross platform development. But before we look at what cross platform development means, we will investigate further what the difference between Android, iOS and Windows Phone are and how one develops native applications to each one of them.
2.1.1 Android

Android is an open source mobile operating system which was originally created by Android Inc, a company acquired by Google in 2005. Google is now developing Android OS together with the Open Handset Alliance (OHA) which is a businesses alliance created for the purpose of developing open mobile device standards [41]. OHA have today 84 member companies, including Qualcomm, HTC, LG, Intel and Google.

Android Operating System is based on the Linux Kernel and consists of an applications written in Java running on top of the application frameworks and the Dalvik Virtual Machine and they uses libraries written in C/C++ [40], a more detailed description of the Android software stack can be seen in figure 1 below.

Figure 1: The Android Software Stack [40].

- **Linux Kernel**
  The bottom layer is the Linux Kernel, which means that the whole Android OS is built on top of the Linux Kernel. The Linux kernel interacts with hardware and contains all essential hardware drivers like camera, keypad, audio etc..

- **Libraries**
  Next layer is a set of libraries including SQLite which is the database engine used to store data, WebKit which is a web browser engine, a media framework that provides recording and playback of media etc..

- **Android runtime**
  Android runtime provides the Dalvik Virtual Machine and Core Java libraries the libraries enable developers to write Android applications using the Java programming language.

- **Application frameworks**
  In this layer many high-level services that an application directly interacts with is located. Examples of such services are the Activity Manager which manages the applications lifecycle and the activity stack, the Resource Manager which provides access to various resources like strings, color settings, etc. in our application and the Location Manager that manages the location using GPS or network.

- **Applications**
  Applications are the top layer of the android architecture. All applications will be in-
stalled on this layer. Example of applications are dialer, web browser and the contact manager.

To develop applications to Android the Android SDK is used, which contains a number of developer tools such as a debugger, libraries, emulators, documentation and tutorials. The Android SDK is today supported on several platforms such as any modern desktop Linux distribution, Mac OS X v10.4.8 or later and Windows XP or later. The officially supported IDE is Android Studio[2].

2.1.2 iOS

iOS is a OS developed by Apple for Apple-manufactured devices, it was originally called the iPhone OS but was renamed to iOS in 2010. iOS is a stripped-down version of the OS X operating system [43]. As seen in figure 2 below there are four abstraction layers within iOS.

![Figure 2: iOS Software Layers [8]](image)

- **Core OS layer**
  Contains low-level features that most other frameworks is built upon, unless you are working with security or interaction with external hardware, then you will use frameworks directly from this layer.

- **Core Services layer**
  Contains the fundamental system services required by upper layers. Such as SQLite, location frameworks and access to the iOS Address Book.

- **Media Services layer**
  Provides frameworks for handling graphical and audio technologies. These frameworks allow developers to create applications that provide graphical interfaces.

- **Cocoa Touch layer**
  Contains the frameworks that normally is used for building iOS applications. Examples of frameworks are the UIKit Framework that provides the infrastructure needed to construct an iOS application and the Message UI Framework which is needed for creating and sending emails and SMS from within an application.
To develop applications for iOS the *iOS Software Development kit* (SDK) which is only supported on a Mac running Apple’s OS X as an operating system. The SDK in combination with Apple’s officially supported IDE, X-Code, contains all tools and interfaces needed to develop, run and test native iOS applications. Native iOS applications is written using Objective-C or Swift [8].

### 2.1.3 Windows Phone

Windows Phone OS launched in 2010 and is an operating system developed by Microsoft, it is the successor to Microsoft’s initial mobile OS, Windows Mobile. In figure 3 below is an illustration over the software layers in a Windows Phone 8.1 application.

![Windows Phone 8.1 Software Layers](image)

- **Windows Core OS Services**
  Widows Phone 8.1 uses the same kernel as Windows 8, namely the Windows NT core.

- **WinRT APIs**
  WinRT is a subset of the full Windows API and provides an API used for accessing features like media, hardware drivers and other system services.

- **Top Layer**
  Visualises what programming languages that can be used for developing Windows Phone applications. Any of these languages can be used to communicate to the WinRT API.

In order to develop a native Windows Phone 8.1 application, the Windows SDK for Windows 8.1 is needed. This SDK includes application templates, debuggers, emulators and more. The official IDE for Windows Phone development is Microsoft’s Visual Studio. You can choose to develop your native application in either C/C++, C#/VB or using JavaScript. The user interface can then be built using ether XAML in combination with C++ and C# or HTML and CSS with JavaScript [31].
2.2 Cross-platform development

Now when we have taken a closer look at the three major mobile operating systems of today we will look at what approaches there exists when it comes to building cross-platform applications that can run on all of these operating systems using one single codebase.

Wikipedia defines the term cross-platform as: "an attribute in computer software or computing methods and concepts that are implemented and interoperate on multiple computer platforms" [46]. Cross-platform mobile application development is nothing new. In fact when the first iPhone was released in June 2007 it did not support any native third-party applications at all, so in order to support the new iPhone a web application had to be developed. This because according to Steve Jobs, there were no reason for developing native applications to the iPhone:

"The full Safari engine is inside of iPhone. And so, you can write amazing Web 2.0 and Ajax apps that look exactly and behave exactly like apps on the iPhone. And these apps can integrate perfectly with iPhone services. They can make a call, they can send an email, they can look up a location on Google Maps."

- Steve Jobs, when releasing the first iPhone.

But web applications never really took off, and four months after the initial release of the iPhone, Steve Jobs changed his mind and announced that the iPhone SDK would be released in March 2008 [1].

When it comes to cross-platform development for mobile applications there exists several different approaches, the different approaches have been described in articles written by Raj & Tolety [35] and Xanthopoulos & Xinogalos [56]. In this thesis we will focus on the three most popular approaches, namely the following: Cross-compiler approach, Web approach and Hybrid approach.

2.2.1 Cross-compiler approach

Using the cross-compiler approach the developer can write an application in one mainstream language and that applications source code can then be cross-compiled and converted down to native binaries for all operating systems that are supported by the compiler. In the best of worlds, this approach would generate complete native applications from one codebase, but how successful this is depends largely on how efficient and reliable the cross-compiler is [35]. A figure illustrating how this works can be seen in figure 4 below.
It is not unusual that you with the cross-compiler approach need to write some platform specific code for the *user interface* (UI), but normally all application logic can be shared across platforms. A Cross-compiler tool normal binds their API to native API’s, therefore the ability to access device specific hardware is high but this also means that for each update of a platforms native API, the tool vendor also need to update their API, in order to support the latest functionality. This extra layer between native and your application can therefore cause some delays in development [35].

### 2.2.2 Web approach

The Web approach is to create a website with standard web technologies like HTML5, CSS and JavaScript and then adjusts and optimize it for mobile devices. There exists many JavaScript/HTML5 libraries/frameworks that will help with the optimization for mobile devices, *jQuery Mobile*, *React.js* and *Ionic* just to mention a few. These libraries makes it possible to handle finger gestures such as tap, pinch and zoom and provides developers with mobile specific user interface components like list views, side panels and toolbars in order to ease the interface development.

When the HTML5 standard [44] was released in October 2014, web applications became enhanced. Since the new HTML5 standard provides additional capabilities when it comes to optimize for mobile devices, like making it possible for developers to directly access some device specific hardware such as GPS, microphone, camera, etc. but also support for off-line storage was added with a simple *key-value* based database just to mention a few enhancements [10, 27].

A mobile optimized website can be displayed in any modern mobile device that has a browser that supports HTML5. And since the application is executed in the browser it will be completely platform independent, but it will still have some limitations when it comes to access device hardware. Web applications cannot usually be installed physically on a device, but are instead downloaded from a web server, therefore there exists situations where the application will be inaccessible to the end user, for example when the device has no service [56].
2.2.3 Hybrid approach

The hybrid approach is a combination of web- and native applications, they are built using technologies like HTML5, CSS3 and JavaScript but are embedded in a native container by using a framework like Apache Cordova or PhoneGap. Since the web application is placed in a native container, the application will need to be installed on the actual device [56].

The native container is simply a native WebView, this means that for Android, the `android.webkit.webview` is used and for Windows Phone the WebView-class is used etc. The WebView normally covers the entire screen of the device. Device capabilities are accessible in the WebView through an abstraction layer from the framework used to generate the native container [35].

![Architecture of a hybrid application](image)

**Figure 5:** Architecture of a hybrid application [37].

In figure 5 above is an illustration over the architecture of a hybrid application when using the Cordova framework. Where the **rendering layer** is a WebView and the Camera JS, File JS etc. are **plugins** to Cordova and trough the Cordova bridge, also known as the abstraction layer, we can access the native implementation of that particular feature.
2.3 Cross-platform development tools

There exists many different cross-platform development tools and frameworks that utilizes the approaches mentioned in the sections above. In the table below are some of the more popular tools of today listed, sorted after what approach they utilizes.

Table 2: Popular tools/frameworks used for cross-platform development.

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Tool/framework</th>
<th>Approach</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoboVM AB</td>
<td>RoboVM</td>
<td>Cross-compiler</td>
<td>Application framework</td>
</tr>
<tr>
<td>Xamarin</td>
<td>Xamarin</td>
<td>Cross-compiler</td>
<td>Application framework</td>
</tr>
<tr>
<td>Adobe</td>
<td>Adobe AIR</td>
<td>Hybrid</td>
<td>Mobile development framework</td>
</tr>
<tr>
<td>Apache</td>
<td>Cordova</td>
<td>Hybrid</td>
<td>Mobile development framework</td>
</tr>
<tr>
<td>Intel</td>
<td>Intel XDK</td>
<td>Hybrid</td>
<td>Mobile development framework</td>
</tr>
<tr>
<td>Nitobi</td>
<td>PhoneGap</td>
<td>Hybrid</td>
<td>Mobile development framework</td>
</tr>
<tr>
<td>Rhodes</td>
<td>RhoMobile</td>
<td>Hybrid</td>
<td>Mobile development framework</td>
</tr>
<tr>
<td>Drifty Co.</td>
<td>Ionic</td>
<td>Web</td>
<td>HTML5/JavaScript Framework</td>
</tr>
<tr>
<td>Facebook</td>
<td>React.js</td>
<td>Web</td>
<td>JavaScript library</td>
</tr>
<tr>
<td>Google</td>
<td>GWT mobile</td>
<td>Web</td>
<td>Ajax framework</td>
</tr>
<tr>
<td>Telerik</td>
<td>Kendo UI</td>
<td>Web</td>
<td>JavaScript library</td>
</tr>
<tr>
<td>Sencha</td>
<td>Sencha Touch</td>
<td>Web</td>
<td>JavaScript library</td>
</tr>
<tr>
<td>jQuery Project</td>
<td>jQuery Mobile</td>
<td>Web</td>
<td>JavaScript library</td>
</tr>
<tr>
<td>Trigger.io</td>
<td>Trigger.io</td>
<td>Web</td>
<td>HTML5 framework</td>
</tr>
</tbody>
</table>

To summarize, there are some great differences in architecture between the three major mobile operating systems of today. But despite that there exists a couple of different approaches that makes it possible to overcome the complication of having to develop a native application to each one of them. But there does not exists one tool that rules them all, instead there exists a growing spectrum of tools that utilizes these approaches. So how should one know what tool that is best suited for development of our application? This is what we hopefully will straighten out in this thesis.
3 Problem Description

In this chapter we will expand the problem statement and look at what cross-platform development tools we will evaluate in this thesis and how they actually work. Lastly we will also state the method that will be used for evaluating the selected cross-platform development tools.

3.1 Problem statement

As mentioned Vitec have a mobile application called Teknisk Förvaltning, which is available on both Android, iOS and Windows Phone 8.1. Teknisk Förvaltning is used as a tool for cost-effective management of the dwelling stock. It can be used to make inspections of a property, perform work orders, do meter readings and perform preventive maintenance.

The application is today built using the JavaScript library jQuery and the framework Cordova. It was first released in June 2011 and have today about 2000 users. The main issues with the application according to developers at Vitec is that since it was first released, new features and functionality has been added gradually, and as a result of using JavaScript, which is a dynamically typed language, it becomes difficult for a developer who does not have full knowledge of how the code works to implement new functionality. This results in when new functionality is actually added, the code reuse of old methods and functions is low and therefore the code have become very messy, unstructured and difficult to read over the years. Another problem mentioned is that the impression of the interface is quite flat. The application currently lacks completely the native-feeling that one wishes to achieve, in addition, the application feels slow when switching between views and navigating through
the application.

Vitec is therefore interested in finding out what other tools there exists when it comes to
developing cross-platform mobile applications and which of these tools are best suited for
them. In this thesis, we will therefore evaluate a couple of the tools listed in table 2 under
chapter 2. The evaluation of the tools will be preformed by first developing an evaluation
framework and by using that framework we will be able to determine which of these tools
that is the most appropriate choice for their application. Hence the problem statement is:

- Can we develop a generic evaluation framework that can be used to asses any cross-
  platform mobile development tool, with the aim to help developers select the most
  appropriate tool for a specific mobile application.

3.2 Cross-platform tools and frameworks we will evaluate

What tools we have chosen to evaluate in this thesis was selected and determined in consul-
tation with developers at Vitec. We decided to look at the following two tools: Xamarin and
Cordova + Ionic. Where Xamarin utilizes the cross-compiler approach and Cordova + Ionic
the hybrid approach. We have decided not to look at any pure web approaches since Vitec’s
application uses NFC, which is not possible when making a pure web application. A native
Android application will also be developed and used as a baseline when evaluating the two
cross platform development tools. The reason we have chosen Android as the platform to
develop a native application for is because most of the users of the current application is
using devices running Android.

3.2.1 Xamarin

Xamarin is one of the tools we will evaluate, it utilizes the cross-compiler approach and
enables developers to build native applications for iOS, Android, and Windows Phone with
a native looking interface using one single language, C# [54]. Xamarin provides two tools
for developing cross-platform applications namely Xamarin.Android and Xamarin.iOS they
are both built on top of Mono, which is an open source implementation of Microsoft’s .NET
Framework. Xamarin was first released in 2004. When it comes to Windows Phone, no
Xamarin product is need to build Windows Phone applications, this is because since the
project already is developed using C#, which is the native language for Windows Phone,
the code be used directly to build an application for Windows Phone 8 and 8.1.

Using the Xamarin platform you can, as a developer, share an applications logic between
platforms but you still have to develop a separate user interface to each platform and write
some platform specific C# code for functionality that is requiring device specific features
like reading and writing NFC-tags. This since NFC capabilities only exists on some Android
and Windows Phone devices and not on any iOS devices [50].
In figure 7 we can see what type of code that can be shared using Xamarin.iOS and Xamarin.Android. According to the developers at Xamarin around 75% of the code, depending on platform, can be shared using this strategy [54].

Xamarin have 100% API coverage [54], this means that any API features that you can access through native development you can access through C# and Xamarin. This is because Xamarin generates a native binding or a mapping of the call from the C# library to the native library, this entails that all API calls actually uses native APIs [54].

The way Xamarin produces native applications differ between the platforms. On iOS an ahead-of-time (AOT) compiler from the mono framework compiles the Xamarin.iOS application and produces a native ARM binary that can then be executed on the iOS platform. The .NET framework is included within the application, but unused classes are stripped out in order to reduce file size [52]. When developing iOS applications with Xamarin a Mac running OS X is required and needed for building and licensing purposes.

On Android the Xamarin.Android application is compiled to Common Intermediate Language (CIL) and unused classes from the .NET Framework are here also stripped out. When the application launches Just-in-Time (JIT) compiling is used to produce a native assembly. In order to develop an Android application using Xamarin both Java and the Android SDK are required.

Using the approach mentioned above thus results as in that about 75% of the code can be shared between the different platforms. But in this thesis we are more interested in a full cross-platform solution, where almost all code can be shared between platforms. For this there exists something called Xamarin.Forms [55]. Xamarin.Forms is a user interface (UI) framework for creating a native user interface that can be shared across Android, iOS and Windows Phone applications.

![Diagram](image.png)

**Figure 7:** Code shared between platforms [54].

![Diagram](image.png)

**Figure 8:** Code shared between platforms using Xamarin.Forms [54].
By using Xamarin.Forms up to 100% code reuse can be achieved [54]. Xamarin.Forms binds at runtime controls and pages to platform-specific native equivalent UI elements. This toolkit is suited for applications that require little platform-specific functionality like network and sensor access and where code-sharing have a higher priority than a highly polished and complicated UI. Therefore since we in this thesis is going for full cross-platform development and since we have an application with a not so complex UI we have chosen to use Xamarin.Forms.

3.2.2 Apache Cordova/PhoneGap

PhoneGap was an open source project started and developed by Nitobi and is a framework that has the function to work as a native wrapper. Nitobi was in 2011 acquired by Adobe, but Adobe decided to donate the original PhoneGap source-code to the Apache Software Foundation (ASF) under the name Apache Cordova, to ensure that it remained open source. Today both PhoneGap and Apache Cordova exists where PhoneGap is more like a distribution of Apache Cordova. According to PhoneGap you can think of Cordova as the engine that powers PhoneGap.

Cordova provides a set of JavaScript APIs which enables a developer to build a website using HTML5, CSS3 and JavaScript and trough Cordovas APIs access native-specific functions like GPS, camera and network. The webiste is then packaged as a native application, this is achieved by using an embedded native WebView that covers the entire screen of the device. The WebView that is used is the same as used by each platform, that means for Android android.webkit.webview is used and for Windows Phone the WebView-class is used etc. [15].

When developing a Cordova application we have a config.xml-file which is located at the root of the project, this global configuration file specifies the applications preferences and behaviours, such as how orientation shifts a similar should be handled. The application itself is nothing but a local web project located in the www-folder in the solution. By default the main page is the index.html-file, that file in turn references any other resources that are needed [13]. This web application is then, as described above, placed and presented in an native WebView on each supported platform.

3.2.3 Ionic

Ionic is a open-source HTML5 framework for mobile application development that was created by Drifty Co. in 2013, a beta was released in March 2014 and a final 1.0 version was released in May 2015 [24]. Ionic offers mobile-optimized HTML, CSS and JavaScript components and is built on top of AngularJS, which is a JavaScript framework for developing dynamic web applications [2]. You can say that Ionic provides similar functionality to AngularJS as the Android UI elements provides for Java. It simply enables developers to focus on building applications instead of building user interface elements and components. Some of the core mobile UI components that are available to developers out of the box when using Ionic is list views, spinners, tab bars, action sheets etc. [19].

Ionic compatibility starts at iOS 6 and Android 4.1, older versions than that will not be officially supported. But since Ionic basically is a mobile website it will probably run in all modern browsers. The Ionic Framework also provides a useful command line interface
(CLI) that makes it easy to start, create, compile and export mobile applications.

The framework provides some useful functions such as `ionic.Platform.isIOS()`, `ionic.Platform.isAndroid()` and `ionic.Platform.isWindowsPhone()` which can be used to detect on what OS the current application is running on. This is useful for example if one wishes to assign different icons depending on what OS the application is running on. An example code of this shown below.

```javascript
var icon;
if (ionic.Platform.isIOS()) {
    icon = 'ion-ios-arrow-back';
} else if (ionic.Platform.isWindowsPhone()) {
    icon = 'ion-windowsphone-arrow-back';
} else {
    icon = 'ion-android-arrow-back';
}
```

### 3.3 Methods

In order to find out what cross-platform development tool that is the most appropriate choice for Vitec, an evaluation framework will be developed. The initial evaluation framework proposal will consist of a set of criteria or decision factors who we in consideration with developers of the current Teknisk Förvaltning-application, consider to be important. These initial criteria will also be based on experience gained when developing the prototype applications. The initial evaluation framework will then be refined by studying scientific papers and articles dealing with similar topics and based on their conclusions and methods we will refine our framework.

In order to evaluate some of the criteria in the evaluation framework, prototype applications built using the different tools we will evaluate, namely Xamarin and Cordova+Ionic, needs to be developed. The prototype applications will be a subset of the current Teknisk Förvaltning-application. Also a native equivalent for the Android platform is to be developed and will be used as a reference and a baseline when evaluating some of the criteria from the evaluation framework. Lastly our findings and conclusions from the evaluation of the cross platform development tools will be summarized and presented in a result table and from that table, and by allowing Vitec to rank the importance of each criteria, we will be able to determine what the most appropriate tool for Vitec is.
4 Evaluation Framework Proposal

The evaluation framework that will be used to assess the various tools mentioned in chapter 3, have been evolved through discussions with developers at Vitec, in order to capture what they consider to be important decision factors but also through experience gained when developing prototype applications in combination with literature studies in order to cover common issues with cross-platform development.

The evaluation framework will be structured and divided into two parts. Where the first part of the framework consist of a comparison of properties between the different tools, the properties we will look at are the following: License costs, Developer(s) of the tool, when it was Released, in what extent can you Access native API’s and what Programming language that is used for development.

The second part of the framework will have more substance, it will be divided into two sections, where the first section will evaluate a set of criteria more from a development perspective and the second section will evaluate criteria from an user perspective.

4.1 Development perspective

First thing we will look at with a development perspective is how well the documentation and support is for the different tools. This criterion will be somewhat based on a subjective assessment of how the documentation and support was perceived when developing the prototype applications. But it will also be based on how many questions and answers to questions there exists on online developer forums like Stackoverflow, since with more questions asked, you will be more likely to find someone who have had, and hopefully solved, a problem similar as yours. The documentation and support criterion is important since with good documentation and support the development of the application will be faster since problems and difficulties that occurs under development will be more likely easier to solve.

Maintainability is another criterion we will evaluate. Maintainability can according to the IEEE Standard Glossary of Software Engineering Terminology be defined as: "Maintainability, […] The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.” It is known that large codebases is hard to maintain we will therefore use "Source lines of code (SLOC)” as a metric. Meaning that we will count the number of lines containing actual code in each prototype application and compare these numbers against each other. There also exists some ways to keep a large codebase somewhat easier to maintain, Gustavo Vale et al. have suggested some guidelines for improving the maintainability of such systems. We will therefore see how well the code developed using our tools will be able to fulfil some of these guidelines. Hence we will look at what support there exists for modularization, encapsulation and re-use of code using the different tools. We describe further
what these three concepts imply [42]:

- Modularization: To what extent can code be factored into modules where every module can have a clear responsibility.

- Encapsulation: Does it exist a mechanism for restricting access to some components / objects? The goal here is to reduce the coupling.

- Re-use: To what extent can code be re-used or inherited. Since if a problem is solved correctly once and can be re-used or inherited in the system, it will be easier to maintain then if a similar functionality need to be rewritten multiple times.

Note that there exists many more evolved metrics than SLOC for measure software complexity [45], but since our prototype applications are written using different approaches and tools it will be hard to get any valuable information from these more advanced metrics.

Development speed, evaluates how easy and fast an application can be developed using the different tools. The metric we are going to use is partly how fast the prototype application could be developed but also how steep the learning curve is for each tool and to what extent for example UI libraries and UI builders or similar are available to ease development.

Build-debug-test, we will also look at what possibilities there exists when it comes to build, debug and test the application. This criterion will be evaluated based on the experience gained when developing the prototype applications.

### 4.2 User perspective

The evaluation framework will also look at some criteria from the user perspective. One of these, and maybe the most important when it comes to the user perspective is the User Experience and the User Interface. This criterion will be quantified by allowing a small number of test persons try each one of the prototype applications and ask them to evaluate them in terms of intuitiveness, recognition factor, look and feel compared with the native developed equivalent where possible and with a native application from another context where no native equivalent were developed, i.e. on Windows Phone and on iOS. For each of the four factors the test persons will be asked to grade the fulfilment of that factors from 1 to 5 and leave a comprehensive comment. The test person will not know which application that was developed using what tool.

Another criterion we will evaluate is Launch time. According to a survey conducted by Zogby Analytics on behalf of CA Technologies [39] showed that if an application is not loaded within 3 seconds, 37% of all users will leave the applications and sometimes never come back. Therefore we will look at how long the loading time is from that the user launch the application to it is fully started.

We will also look at deployment. How is the application distributed to end-users? And how easy is it deploy and distribute the application using those channels?
5 Evaluation Framework Enhancement

There exists a couple of related evaluation frameworks for evaluating cross platform development solutions. These related frameworks have varying quality but some of them will be mentioned in this chapter. We will focus on what methods they have used and what the result and their conclusions of using those methods were.

5.1 Related evaluation frameworks

There exists a few articles written by Palmieri et al. and Ribeiro et al. [32, 36], who have made more of a factual comparison of different cross-platform tools, i.e. they have not made any actual prototype implementations. The aim of their work was to help developers to select the appropriate tool that matches their requirements. Their approach were to select a few, at the time, popular cross-platform development tools, based on the criterion that they should be able to generate applications to all major platforms. Then simply compare properties of the different tools, some of the properties they both articles have looked at are the following:

- Licensing cost
- Programming language used
- Ability to access native API’s
- What platforms they support
- Development environment used

The result of their work is a number of tables where they have evaluated the properties mentioned above for each selected tool. And from that data they have been able to drawn some conclusions of strengths and weaknesses of each tool. They have only evaluated at one of the tools we have selected to assess in this thesis namely PhoneGap/Cordova. The conclusion from the article written by Ribeiro et al. is that PhoneGap is suited to web developers who want their existing web application converted to native in order to distribute them on the mobile application markets [36]. Both articles [32, 36] mentions as a weakness that PhoneGap does not support any official IDE, which according to them means that as a developer you will need to import you code to different IDEs (e.g Android Studio for Android and Xcode for iOS) in order to deploy your application on different platforms [36].

A number of articles have used an approach more like ours [16, 18, 21], meaning that the evaluate tools based on an actual implementation utilizing the different tool. The purpose of all these articles is to, based on some evaluation framework they have developed, help developers selecting the most appropriate cross-platform development tool. They have all
selected tools to evaluate based on the criteria that they should work on most platforms and be extensively used in practice. Non of the papers have evaluated Xamarin, but many of them have looked at PhoneGap/Cordova.

Two of them, Dalmaso et al. [16] and Dhillon & Mahmoud [18], focuses more on what performance we can expect from implementations with the different tools, one by making and evaluating benchmarks applications of the different tools [18], and the other by looking at the memory usage, CPU usage and power consumption of applications implemented with the different tools[16]. But they both also have a section where they look at other decision factors like:

- **Security** - It is of importance for most service providers that their application is secure, with secure means among other things that user data or source code is not leaked from the application.
- **User interface** - They have looked at how similar the UI of cross-platform developed applications are when comparing them with a native equivalent.
- **Access to device API**- Evaluating what possibilities there exists when it comes to interaction with device hardware for each of the tools being evaluated.

The article [16] who have evaluated memory usage, power consumption and CPU usage had the aim to see how well optimized the developed cross-platform applications is compared to native. Their conclusion is that among the tools they have evaluated PhoneGap/Cordova + HTML and CSS was found to have lowest memory, CPU usage and power consumption, but with this solution comes also a very simple user interface. When using PhoneGap + JQuery Mobile to render UI elements in order to enhance the user experience and interface, both memory, CPU usage and power consumption was found to be higher.

The other article [18] have tried to benchmark cross-platform applications and have focused on two types of benchmarks, processor intense and data intense. For processor intense they have implemented and evaluated AES encryption and Input validation, for data intense they have benchmarked: sorting algorithm, remote service access and local PIM access. The result of their work is not unequivocal, no clear winner cloud be crowned. Depending on what type of application there is to be developed, different tools should be used, for example: for a simple applications PhoneGap should be used but for applications with a more advanced UI, Titanium should according to the authors be used, when looking at the tools they have evaluated. Their overall conclusion is that cross-platform development tools generally show performance issues when they are compared to their native equivalent.

A paper written by Henning Heitkötter et al. [21] is the paper who was most similar to outs, they have also the aim to develop an evaluation framework to assess cross-platform development tools. Based on this evaluation framework they have then evaluated applications developed with PhoneGap and Titanium Mobile. For comparison they have also looked at native developed applications. Their evaluation framework have emerged from discussions with different software companies, from literature research, online developer communities and from experience gained when developing prototype applications. The evaluation framework list 14 criteria, structured into infrastructure and developer perspective. The criteria are listed in the table below.

The result of their evaluation of the cross-platform tools: PhoneGap and Titanium Mobile, is that PhoneGap is among these the preferred tool when developing cross-platform appli-
Table 3: Evaluation framework by Henning Heitkötter et al. [21]

<table>
<thead>
<tr>
<th><strong>Infrastructure Perspective</strong></th>
<th><strong>Developer perspective</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>License and costs</td>
<td>Development environment</td>
</tr>
<tr>
<td>Supported platforms</td>
<td>GUI Design</td>
</tr>
<tr>
<td>Access to platform specific features</td>
<td>Ease of development</td>
</tr>
<tr>
<td>Long-term feasibility</td>
<td>Maintainability</td>
</tr>
<tr>
<td>Look and feel</td>
<td>Scalability</td>
</tr>
<tr>
<td>Application Speed</td>
<td>Opportunities for future development</td>
</tr>
<tr>
<td>Distribution</td>
<td>Speed and cost of development</td>
</tr>
</tbody>
</table>

cations, unless native-looking user interface is of high importance then Titanium Mobile should be considered.

5.2 Enhancement of our evaluation framework

From the related evaluation frameworks mentioned in previous section we have learned that there exists many ways to evaluate cross-platform development tools. We will now based on these related evaluation frameworks and their conclusions improve our evaluation framework.

One criterion all of the papers [32, 36, 16, 18, 21] have chosen to evaluate is what integrated development environment (IDE) that each tool utilizes, this because if there do not exist a official IDE, development will be more complicated since you will need to import your code to different IDEs e.g Android Studio for building Android application and Xcode for iOS applications[36]. Therefore we will add Integrated development environment to the first part of our evaluation framework where we look at different properties of each tool.

In the paper written by Henning Heitkötter et al. [21] they have a criterion called Opportunities of further development where they evaluate the risk of vendor lock-in which occurs when we are unable to switch to another approach without a substantial rewriting of source-code. This criterion cloud be of importance, especially if you have an application that is supposed to be used “forever” and then one day the developing of the tool stops, then you are stuck with a tool that may not longer support the latest operating system and features and you are needed, at a high cost, switch to another approach. Therefore Risk of vendor lock-in is added to the first part in the evaluation framework.

Both Dalmasso et al. [16] and Dhillon & Mahmoud [18] have evaluated the user interface and the user experience of their developed prototype applications and have as a metric compared their applications with a native equivalent. We have in our original evaluation framework chosen to evaluate the user interface and the user experience by letting a group of test persons try each implementation and grade them from a number of factors. But to get a bigger picture we will also look at some components in the interface and compare them with the same component in a native developed application to get a feeling of how similar to native the cross platform applications UI are.

The two papers written by Dalmasso et al. [16] and Dhillon & Mahmoud [18], who have evaluated cross-platform development tools partly by implementing benchmarks and partly
by analysing the memory- and CPU usage of their developed prototype applications. Have from this come to the conclusion that there exists performance issues when they compared their cross platform implementations to their native equivalent. We have therefore chosen to investigate this further and see if that is the case when we measure the memory- and CPU usage on our cross-platform implementations too. We will therefore add the criterion Memory usage and CPU usage to part 2, section 1 in our evaluation framework.

The memory comparison will be done in a similar way as Dalmasso et al. [16], this includes that we will only preform memory analysis on the Android platform and the analysis will be done by comparing the Proportional Set Size (PSS) and Unique Set Size (USS) of each implementation. Where the value PSS is a metric the kernel computes from how much RAM that is allocated for a process, this includes both dirty and clean pages and pages that is shared with other processes [38]. But the memory that is shared is evenly distributed between the processes who share that memory, so that means that if 10 MB is shared between two processes each process gets 5MB to their PSS value [4]. USS or also known as Private Dirty is the most expensive RAM, since it is the RAM inside a process that can not be be shared with any other process, and its contents exist only in RAM so can not be paged to storage [38]. To analyse the RAM allocation of on an Android Device the following Android Debug Bridge (ADB) command is used:

```
adb shell dumpsys meminfo <pid>
```

Where `<pid>` is the process id of the process whose memory we want to analyse.

For analysis of CPU usage in our prototype applications we will use the same approach as Dalmasso et al. [16] did. This means that we will perform the analysis on implementations running on the Android platform only and we will use the `top` command:

```
adb shell top
```

To get information on the CPU usage in our applications. The output of the top-command is a snapshot of the CPU load at that particular moment, so note that these values may vary from a millisecond to another. We will using this command to figure out where each prototype application does the most work, i.e. have the highest CPU load, and compare this number with the other implementations. From this we will come to a conclusion of what implementation who is the most CPU efficient.
5.3 Refined evaluation framework

From the improvements of our proposal evaluation framework made in the section above, we have developed a refined version of our evaluation framework presented initially in chapter four. The new refined version is visualized in table 4 below.

Table 4: Final version of our evaluation framework

<table>
<thead>
<tr>
<th>Evaluation Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
</tr>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>Developer</td>
</tr>
<tr>
<td>First Released</td>
</tr>
<tr>
<td>IDE</td>
</tr>
<tr>
<td>Licensing cost</td>
</tr>
<tr>
<td>Device API access</td>
</tr>
<tr>
<td>Risk of vendor lock-in</td>
</tr>
</tbody>
</table>

The method we will use for evaluating cross-platform development tools with this framework is to first evaluate part one of the framework by comparing the properties against each other and against native development. Then under section one and two in part two of the framework we will evaluate the criterion using experience gained when developing and analysing the prototype applications and from reading documentation about the different tools. Where possible, we will use the native developed prototype application as a baseline in our evaluation. We will also grade the fulfilment of each criterion under part 2 of the framework in a scale from one to five, where five (5) is best/high/easy/efficient and one (1) is worst/low/hard/not efficient. You can also say that a five is equivalent to native, meaning it is as good as it gets.

When the evaluation is complete, we will summarize the result in a table. In that result table we will list the properties from having completed part 1 of the evaluation framework and summarize the grades from the fulfilment of part 2 of the framework. Using this result table a developer can then from whatever aspects and requirements they consider to be important, choose a cross platform development tool that fulfils these requirements and thereby choose the most appropriate tool that fits their needs.
6 Evaluation

This chapter will begin with a presentation of the developed prototype applications. Then the evaluation itself will be done using the final evaluation framework and approach that was presented in chapter 5, using a native developed Android application as a baseline. This means that in each criterion where it is applicable to compare our cross platform implementations with a native equivalent we will do so.

When all criterion in the framework have been evaluated we will summarize our findings in a result table by stating the properties found in part 1 of the framework and by grading the criteria from part two of the framework from one to five, depending on fulfilment.

6.1 Prototype applications

The prototype applications consists, as mentioned earlier, of a subset of Vitec’s mobile application Teknisk Förvaltning. The original application consists of four modules, we will in our prototype application implement one of those modules, namely the module ”mätaravläsning” which is used to do energy monitoring in buildings. A property owner or similar uses the module ”mätaravläsning”, which contains a list of all electricity- and consumption meters available in a property portfolio, to manually preform and save a reading of a meter. This is normally done once a month. The data from these preformed meter readings are then used to calculate costs for that property or building. The reason that we have chosen to implement the module ”mätaravläsning” is because no other module contains any additional functionality in terms of access to device hardware than the module we have chosen to implement.

The prototype application consist of a log in screen where REST and JSON is used to communicate with a Windows Communication Foundation (WCF) service in order to authenticate a user. Once a user is authorized all meters who is register to that user is fetched from the same WCF Service, these meters which comes as a JSON are then parsed and presented as a list within the application. That list is also sortable on variables like entity name, meter name, read or unread etc.. A user can click on a meter to get to a detail-page for that particular meter. A user can also use the built in barcode scanner to scan a barcode and if that barcode is assigned to a specific meter that meters detail-page is shown on the screen, if no meter is linked to that barcode the user can choose to assign it to a meter of his or her own choosing. That meter is then updated and the new data saying that this meter now have a barcode assigned to it is posted to the WCF service. Another way to access a detail-page is to use the NFC capabilities of an Android or Windows Phone device. If a NFC-tag is scanned and have a payload that corresponds to an unique id of a meter, then the detail-page of that meter is shown on the screen. If the NFC-tag is empty, or contains data that does not correspond to a meter, the user can choose to write the unique id of a meter of his or her own choosing to the NFC-tag. So next time that NFC-tag is scanned the
The detail-page of a meter itself is nothing but a table where all data linked to a meter is presented, some fields like consumption (förbrukning) and comment (kommentar) are needed to be filled in when a meter reading is performed. Some types of meters need to be replaced now and then and therefore there is also an option to perform a so-called meter-change, when performing a meter-change the user needs to fill in a couple of more entities like date of the change, stop- and start value of the meter and a meter constant. If all data entered by the user is correct that meter's data is parsed to a JSON and posted to the WCF service using REST. But since we can not rely on that the user always has an active internet connection, we need to store updated meters on the device and send the data to the WCF service as soon as we have an active internet connection.

Three different prototype applications have been developed where two is a cross-platform solution and the third a native Android application developed in Android Studio on a Windows 8.1 machine. In figure 9 below are some of the views from the Native developed application presented.

The hybrid application using Cordova and Ionic is developed in Visual Studio 2015. On a Windows 8.1 machine and on a MacBook Air running latest version of OS X namely "El Capitan" for building iOS applications. In figure 10 below are some views of that application presented, since this application is a hybrid application and therefore looks the same on all platforms we have chosen to only show views when the application runs on an Android device.
Figure 10: Hybrid application using Cordova + Ionic running on Android.

Lastly the cross-compiler solution which was developed using the Xamarin platform and with the UI toolkit Xamarin.Forms it is also developed using Visual Studio 2015. The views of the implemented Xamarin.Forms application running on the three different platforms are shown in figures 11-13 below.

Figure 11: Xamarin.Forms application running on Android.
6.2 Part 1 - Properties

We start our evaluation by finding the properties stated in part 1 of the framework for each tool we have chosen to evaluate. We also take a look at properties of the native alternatives.

Native

As already mentioned the three major mobile platforms of today are: Android, iOS and Windows Phone, developed by Google, Apple and Windows respectively. They all support their own official IDE: Android supports Android Studio, iOS Xcode and Windows Visual Studio. There exists no licensing cost for developing and testing native applications to any of the platforms. However exists a cost if you want to distribute your application on respectively platforms application store, for iOS that cost is $99 USD per year for an individual and for Android it is $25 USD per year and for Windows Phone $19 USD. This is a cost you will need to pay to publish your application on their application store, regardless if your
application is native, hybrid or cross-compiled.

**Xamarin**

Xamarin Platform is developed by Xamarin and was first released in February 2013. Xamarin supports both Microsots Visual Studio and their own IDE Xamarin Studio. The licensing cost for Xamarin is $999/year per platform per developer, meaning that for developing to both iOS and Android the yearly cost is $1,998. But there exists a starter edition which allows you to deploy your application to all platforms using Xamarin Studio, but with the restriction that the developed application must meet a size restriction of only 128kB compiled C#-code [51].

Using Xamarin you will have 100% API coverage, meaning that everything you can access when developing native you can access using Xamarin. There is always a risk of vendor lock-in when using a third-party developed tool but Xamarin uses C# for development and so does Windows Phone too, so there will be at least some code-reuse possible.

**Cordova + Ionic**

Cordova is currently owned by Apache Software Foundation (ASF) but was originally created by Nitobi in 2011 then under the name PhoneGap. Ionic is developed by Drifty co. in 2013. There is no official IDE, any IDE or even a simple text editor, can be used for development. Cordova uses command-line interface (CLI) for building and testing, which allows you to create projects with any IDE you like and then build and test them on devices/emulators using the Cordova CLI. Both Cordova and Ionic is free and have no licensing costs.

Cordova is shipped with a minimal set of APIs, but extra APIs can be added as they are required trough plugins. Some of the APIs available out-of-the-box is: Battery status, camera, contacts, accelerometer, compass and geolocation. A complete list of available APIs is available here [14]. Using Cordova and Ionic there exists is no vendor lock-in, since the developed JavaScript, HTML and CSS-code can easily be reused with another framework.

### 6.3 Part 2 - Development perspective

In part 2 of the evaluation framework we will evaluate each cross platform implementation on every criteria and were possible use the native developed Android application or experience gained from developing that application as a baseline. We start by evaluating from a development perspective.

**Documentation and support**

The quality of the Android documentation is in my opinion very good, clear official documentation, many third-party tutorials, books and code samples are available. The online developer community StackOverflow have 752,552 questions asked with the tag Android. For iOS the same number is 386,477 and for Windows Phone 20,100.

The official documentation form Xamarin is well written but since there exists three differ-
ent Xamarin products in the Xamarin platform: Xamarin.iOS, Xamarin.Android and Xamarin.Forms it is sometimes hard to figure out to what product the documentation is for and if you have found a documentation over something, but for the wrong product it is not clear how, or even if, you can find the same documentation for another product. So the structure of the documentation is something they should improve. There exists some code examples and concept applications written using Xamarin. But the concept applications are messy and hard to follow, they were not of great help when developing the prototype application. Xamarin have a fantastic online developer community on their site, where even developers at Xamarin comes to answer questions. Maybe is it since they have such good community in-house there exists only 9,679 questions tagged with Xamarin on StackOverflow.

The documentation for Cordova and Ionic is good, despite the fact that Ionic is a pretty new framework, was first released in 2013, the official documentation from Ionic is good. They have an excellent starting guide [26] describing step-for-step components in the framework and how they are used to build applications. Ionic also have an in-house online developer community which is the same success story as Xamarin’s forum, plenty of users and quick and detailed answers to questions. At StackOverflow there exists 9669 questions tagged with ionic or ionic-framework. If we look at Cordova it is so easy to use and therefore there was no need of going through the documentation very deeply when developing the prototype application. By going through a simple set-up guide you are up and running in no time. But since Cordova is shipped with a minimal set of API [14] it is the documentation of third party plugins that is more interesting, in the prototype application a plugin called Cordova BarcodeScanner [30] was used for barcode scanning and for that up and running StackOverflow was of great use with almost 40,000 questions tagged with the word cordova many of them dealing with the subject plugins to Cordova/PhoneGap.

To summarize i would say that none of the cross-platforms solutions can match the documentation available for native Android. But in the combat between Xamarin and Cordova+Ionic, especially if we look at the availability of tutorials and questions/answers online, the Cordova+Ionic solution is a clear winner. Therefore Xamarin scores a grade of four, while Cordova + Ionic get three points when we evaluate this criteria.

Maintainability

The native Android application contains around 1,400 lines of actual source code, which is much. But since native Android applications are written using Java, which is an object-orientated language, hence with a good object-oriented code it should be fairly easy to provide a good modularization, encapsulation and it should be easy to re-use code, therefore the application should be fairly easy to maintain despite the fact that it contains more lines of code than the other two implementations.

For the Xamarin.Forms implementation it summarizes up to around 900 lines of actual source code and since C# also is an object-oriented language it should also be easy to maintain and it does provide good support for encapsulation and code re-use. The main reason the SLOC is lesser for Xamarin compared with native android is because it is easier to build UI in Xamarin.Forms, much fewer lines of code is needed to define a UI when using Xamarin.Forms compared with native Android. Our prototype application uses NFC and for that we had to write platform specific code on both Android and Windows Phone. So if an application uses many platform specific features and therefore have a high amount of platform specific code, it will likely become harder to maintain the code since it becomes
more unstructured.

The SLOC for the Cordova+Ionic project sums up to 800 lines. Again the reason that it is that low compared to native Android is mainly because there exists predefined UI components so minimal code is needed to be written in order to produce the UI. Since Ionic uses JavaScript which is a dynamically typed language, this implies that type errors will not be detected by the compiler, these errors are instead detected first at runtime, which in itself makes the code more difficult to maintain since it becomes harder to find errors. Another thing with JavaScript language is that there is no support modularization in terms of classes, namespaces or interfaces. The inheritance system in JavaScript, called prototype inheritance, is in my opinion weak and hard to use. When it comes to encapsulation it does not really exists in JavaScript, properties can be accessible anywhere. These factors makes the Cordova+Ionic project harder to maintain than the other two solutions.

So to summarize, the maintainability for the Cordova+Ionic is under average and therefore achieves a grade of two, while the maintainability for Xamarin is high but since there might be cases where one have to write platform specific code which will decrease maintainability, therefore we grade Xamarin with a four.

Development speed

The prototype Android application took 80 hours to develop, what could have influenced how long it took to develop is that i have some prior Java and Android development, especially in standard Java. With the official IDE for Android development, Android Studio, comes an integrated UI Designer which can be used for building a UI by dragging and dropping components. Using Xamarin.Forms the application took 100 hours to develop. The reason for this was mainly because it was hard to understand how an application in Xamarin.Forms was structured and how the data-binding between a view element and the view model should be implemented. But once one understand how to write a Xamarin application, development should be relatively fast and easy and since Xamarin is built on .NET platform familiar frameworks like Json.NET and similar can be used to ease development [51].

Development of the hybrid application using Cordova+Ionic was the prototype application that took the shortest time to develop, only 70 hours. One reason for this is that all UI components are predefined when using Ionic so building a nice looking UI was extremely easy, but then of course if one would like a more custom looking UI it is possible with some editing in CSS-files at the cost of time. Another thing that made the development time so short was that Ionic together with AngularJS is very effective when it comes to manipulation of the DOM-tree, presenting a complete list of meters in the prototype application on the screen was done by only 4-5 lines of code, compare with the native Android application where over 100 lines of code were required to achieve the same thing. The only drawback i experienced when developing using Cordova+Ionic was that the learning curve was quite steep and since i had no prior experience with angular it took a fair amount of time to understand how to develop using these techniques. But once i got over the learning phase, development was quite straight forward.

The presented time it took to develop each application includes all time it took to learn using the frameworks/tools by doing different tutorials, prototype implementations and reading documentation. In figure 14 below is a graph over the development progress according to
time when developing the prototype applications using the different tools. This data were collected by time tracking how long time it took to develop the application from start to finish and by noting checkpoint times at 25%, 50% and 75% of the development progress. The graph also includes the time it took to learn develop using the different tools, referenced as the learning curve.

![Graph showing development speed of prototype applications](image)

**Figure 14**: Development speed of prototype applications.

As we can see in the figure 14 above, the learning curve between Xamarin.Forms and Cordova+Ionic before development of the prototype application stated are almost identical, but once development started the Cordova+Ionic curve planned out while the curve for Xamarin.Forms continued to increase until we reached 50% of the prototype development progress. The conclusions drawn from this graph is that Cordova+Ionic is a clear winner in the criterion development speed, this mainly because once one understood how to build an application using this solution development was fairly easy and if you already have some knowledge in web development it is no doubt that you will produce a hybrid application using these tools in a short amount of time. Therefore the Cordova+Ionic solution receives the highest grade, a five, when it comes to Xamarin.Forms the development time using Xamarin was longer compared with native Android and thereby Xamarin earns a grade of two when looking at this criteria.

**Build-debug-test**

When debugging a native Android application in Android Studio we make use of the *Dalvik Debug Monitor Service*(DDMS) which is a part of the Android SDK. DDMS comes with a set of tools used for finding bugs in applications running on either an emulator or an actual Android device. One of the tools is *LogCat*, with LogCat you can collect and view system debug output, like log messages regarding the state of the application. Using LogCat you can also see at exactly what line an error have occurred. The DBMS also provides the ability to simulate incoming calls, SMS, and location data spoofing. You can also simulate
different types of networks with different speed and latency such as GPRS, 3G, WIFI etc. When it comes to testing your application Android Studio provides both emulators that utilizes the Android Virtual Device (AVD) configurations and the possibility to preform USB debugging on an actual hardware device using Android Debug Bridge (ADB).

When developing a Xamarin.Forms application in Visual Studio 2015 we make use of the Mono Soft Debugger. The Mono Soft Debugger is running inside the Mono runtime which is different from regular debuggers who normally controls a program without the knowledge from the application that it is being debugged. The IDE, in this case Visual Studio, can then communicate with the Mono runtime and request debugging operations to be performed. You can set breakpoints, preform single stepping and everything else you normally can do with a debugger you can do using Xamarin. One thing i experienced when developing the prototype application was that sometimes i had weird errors that without any good stack trace i.e. the debugger was not able to pinpoint exactly at what line an error occurred. To develop an iOS application you need access to a Mac OS X computer due to Apple licensing, but if you are developing on a Windows machine you can make use of the Xamarin Mac Agent to pair your Windows machine to a Mac over SSH, which then allows you to create, build, and debug iOS applications from a Windows machine. There exist the possibility to deploy and test your application on an emulator running Windows Phone/Android/iOS or a real physical device but process of deploying to both an emulator and to physical device is slow, especially on Android.

The hybrid application using Cordova + Ionic was developed in Visual Studio 2015 and for debugging we used the remote debugging solution from the Chrome DevTools, which allows you to remote debug the WebView of your hybrid application on an Android 4.4+ device or an emulator from the Chrome browser on your developer machine. A similar solution is available if you are using an iOS 6.0+ device, then you can use Sairis Web Inspector tool to remote debug the UIWebView in your hybrid application, but in order to run your application on an Apple device you will need to build your application on a Mac running OS X, this means that if you are not developing on mac computer you will need to move your entire codebase to a mac and Xcode to test your application on an Apple device or emulator. When developing a hybrid application using Ionic you can also choose to test and debug your application directly in a browser, with Ionic it is easy to setup a live-reload server for your project, this means that as soon as any changes are saved to your HTML, CSS, or JavaScript files, the browser is updated with the newest content.

Xamarin debugger have some deficiencies but is overall good, the ability to link a Mac to a Windows machine over SSH is very nice. Xamarin earns a grade of four. Debugging Cordova+Ionic using Chrome DevTools is smooth but it is onerous to have to use different IDEs for building applications to different platforms, therefore Cordova+Ionic is assigned a grade of three.
Memory usage

We will analyze the RAM usage of the three different implementations when running them on the Android platform, this will be achieved by using the adb command: `adb shell dumpsys meminfo <pid>`. Below is an example output of the memory usage for the Xamarin.Forms application from the adb command.

```
> adb shell dumpsys meminfo 30480

Applications Memory Usage (kB):
Uptime: 733602308 Realtime: 1459386764
** MEMINFO in pid 30480 [TekniskForvaltningXamarinForms.Droid] **

<table>
<thead>
<tr>
<th></th>
<th>Pss</th>
<th>Private</th>
<th>Private</th>
<th>Heap</th>
<th>Heap</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Dirty</td>
<td>Clean</td>
<td>Size</td>
<td>Alloc</td>
<td>Free</td>
</tr>
<tr>
<td>Native Heap</td>
<td>15913</td>
<td>15620</td>
<td>0</td>
<td>33280</td>
<td>29955</td>
<td>3324</td>
</tr>
<tr>
<td>Dalvik Heap</td>
<td>24966</td>
<td>24912</td>
<td>0</td>
<td>48346</td>
<td>32093</td>
<td>16253</td>
</tr>
<tr>
<td>Dalvik Other</td>
<td>360</td>
<td>360</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stack</td>
<td>272</td>
<td>272</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gfx dev</td>
<td>27800</td>
<td>27620</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other dev</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.so mmap</td>
<td>2609</td>
<td>156</td>
<td>2072</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.apk mmap</td>
<td>6537</td>
<td>0</td>
<td>6380</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.ttf mmap</td>
<td>63</td>
<td>0</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.dex mmap</td>
<td>1312</td>
<td>4</td>
<td>1308</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.oat mmap</td>
<td>1335</td>
<td>0</td>
<td>364</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.art mmap</td>
<td>1140</td>
<td>776</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other mmap</td>
<td>44</td>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EGL mtrack</td>
<td>39040</td>
<td>39040</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unknown</td>
<td>8456</td>
<td>8456</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>129851</td>
<td>117224</td>
<td>10176</td>
<td>81626</td>
<td>62048</td>
<td>19577</td>
</tr>
</tbody>
</table>
```

We will look at two columns from this data, namely *PSS total* and *Private Dirty total*. The reason we focus on these two types is because PSS is known to be a good measure for the actual RAM weight of a process and USS or Private Dirty is the most expensive type of memory, and are therefore of importance to keep at a low level [3]. We have analyzed the memory usage of the prototype applications, and our findings are presented in table 5 below.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>PSS total (kB)</th>
<th>Private Dirty total (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cordova + Ionic</td>
<td>166 422</td>
<td>143 432</td>
</tr>
<tr>
<td>Native Android</td>
<td>66 567</td>
<td>57 684</td>
</tr>
<tr>
<td>Xamarin.Forms</td>
<td>129 851</td>
<td>117 224</td>
</tr>
</tbody>
</table>
From the above table it is clear that the application written in Cordova + Ionic has the largest PSS and Private Dirty value and Native Android has by far the smallest, more than half of the RAM usage the Xamarin.Forms implementation have allocated. From the same numbers we come to the conclusion to grade Xamarin with a three and Cordova+Ionic with a two when it comes to memory efficiency.

**CPU usage**

We will also investigate the CPU usage of the three different implementations that runs on the Android platform. The metric we will use is the value in the CPU column from the top-command: adb shell top -m 10. Below is an example output from the top-command.

> adb shell top -m 10
User 3%, System 2%, IOW 0%, IRQ 0%
User 49 + Nice 13 + Sys 46 + Idle 1696 + IOW 11 + IRQ 3 + SIRQ 7 = 1825

<table>
<thead>
<tr>
<th>PID</th>
<th>PR</th>
<th>CPU%</th>
<th>#THR</th>
<th>VSS</th>
<th>RSS</th>
<th>PCY</th>
<th>UID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3361</td>
<td>2</td>
<td>8%</td>
<td>S</td>
<td>41 2241280K 121068K fg u0_a108 com.ionicframework</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4885</td>
<td>1</td>
<td>0%</td>
<td>S</td>
<td>109 2471776K 108696K fg system system_server</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>369</td>
<td>0</td>
<td>0%</td>
<td>S</td>
<td>14 148660K 8812K fg system /system/bin/surface</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7072</td>
<td>3</td>
<td>0%</td>
<td>S</td>
<td>41 1647772K 48876K fg u0_a13 com.google.gapps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3461</td>
<td>5</td>
<td>0%</td>
<td>R</td>
<td>1 6000K 1532K fg shell top</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25593</td>
<td>0</td>
<td>0%</td>
<td>S</td>
<td>112 1758096K 77832K bg u0_a134 com.android.app</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>402</td>
<td>2</td>
<td>0%</td>
<td>S</td>
<td>1 0K 0K fg root rcu_preempt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>1</td>
<td>0%</td>
<td>S</td>
<td>1 0K 0K fg root irq/215-fc38800</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7582</td>
<td>0</td>
<td>0%</td>
<td>S</td>
<td>91 2464536K 61832K fg u0_a13 com.android.gms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We have by executing this command, which automatically refreshes the data once every second, been able to determine where the prototype applications does the most work and from that get the highest CPU load each application reaches. Our findings in terms of CPU load are presented in table below.

<table>
<thead>
<tr>
<th>Highest CPU load</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cordova + Ionic</td>
<td>24%</td>
</tr>
<tr>
<td>Native Android</td>
<td>8%</td>
</tr>
<tr>
<td>Xamarin.Forms</td>
<td>11%</td>
</tr>
</tbody>
</table>

From these numbers we can come to the conclusion that the Xamarin.Forms implementation is almost as CPU efficient as the native developed application, while the hybrid application developed using Cordova + Ionic is much less efficient with a 2-3 times higher CPU load than the other two implementations. We can thereby confirm that Dalmasso et al. [16] and Dhillon & Mahmoud [18] conclusions still stands, a hybrid cross-platform application can not compete with a native equivalent in terms of CPU- and memory efficiency, however a cross-compiled solution is getting closer, at least when we look at CPU efficiency. Therefore
the Xamarin.Forms solution will earn a grade four and the Cordova+Ionic solution, which did not perform well on this criteria, will earn a grade of two.

6.4 Part 2 - User perspective

We will now evaluate the criterion in part 2 of the framework. Starting off with the user interface and the user experience.

User Interface and User Experience

We will in this section evaluate the user interface and the user experience of each implemented prototype application. This will be achieved by enabling a number of test persons evaluate each application from a set of given aspects. But we will also compare some UI components from the interfaces against the native equivalent component in order to see how similar our applications look to a native developed application. We will start with the comparison of user interface components against native components.

Comparison of user interface components

We will start or comparison by looking at a component that is on the iOS platform called a Action Sheet [9], on Android a AlertDialog [5] and on Windows Phone a Application Bar Menu [29]. In figure 15 below we can see how the AlertDialog looks in different implementations when running them on an Android device.

The AlertDialog in the Xamarin.Forms application (figure 15b) is identical to the native implementation (figure 15a) while the Cordova + Ionic implementation (figure 15c) have almost no similarities with the other two AlertDialogs, firstly it is placed at the bottom of the screen and not in the centre as the other two and secondly the AlertDialog implemented with Cordova + Ionic has no “ångra”-button (cancel-button) instead you need to touch outside the AlertDialog or press the software back button in order to close it. When it comes to iOS as mentioned earlier the same type of component is called an Action Sheet [9] in figure 17 below is a screenshot of how the Action Sheet look when running the prototype applications on iOS, but since we have not implemented a native iOS application we have taken a screenshot of a native Action Sheet from another context so we have something to compare against.

Figure 15: AlertDialogs on Android.
As seen in the figure above, both figure 16a, 16b and 16c are almost identical the only thing that differ is that the “ångra”-button (cancel-button) is red in the Xamarin.Forms implementation, but that is something you as a developer can programmatically set, a red coloured text in a ActionSheet implies that an option is considered a “dangerous alternative” \[9\]. Lastly we look at the Windows Mobile platform, since we have not implemented any native prototype application on Windows Phone either we have used a native Application Bar Menu from another context for comparison.

Here is where both the Xamarin.Forms and Cordova + Ionic implementation fails. The Xamarin.Forms implementation have the menu on top of the screen and the Cordova + ionic implementation have certainly the menu at the correct position on the screen but the menu have aside from that no similarities with the native Application Bar Menu something that the Xamarin.Forms implementation at least has.

We will also examine the component DatePicker. Date pickers is normally tied to an input field, focus on the input filed and an interactive calendar will open and the user can select a date. We will start by comparing the component on the implementations running on Android Devices.
As we can see in figure 18, the native and the Xamarin.Forms implementation of a DatePicker is identical while the Cordova+ionic implementation also shows up in the centre of the screen but does aside from that not look anything like the native equivalent.

On iOS there exists a small difference between the native iOS DatePicker and the Xamarin.Forms implementation running on iOS, but they are very similar. When it comes to Cordova+Ionic the DatePicker there looks the same as it did on Android.
On the Windows Phone platform the Xamarin.Forms implementation seen in figure 20b looks like on the other two platforms very similar to the native equivalent seen in figure 20a. And the hybrid implementation looks the same as on the other two platforms.

We can already, despite the fact that we only have compared two components, draw the conclusion that the Xamarin.Forms implementation look very close to native on all platforms, while the Ionic interface have more of a static look and does not have that many similarities with a native application in terms of appearance. But you can discuss if it really is desirable to try to achieve a native appearance, there exist something called the Uncanny Valley effect [49], which is a hypothesis identified by the robotics professor Masahiro Mori that states that the closer a robot gets to resembling a human the more humans will like it, but there is a point, just before it looks exactly like a human, where humans become strongly repulsed by the robot. The figure 21 below illustrates the Uncanny Valley concept further.
This hypothesis can according to some \cite{22} be applicable on user interfaces as well, meaning that a user that uses the Xamarin.Forms application, will feel that there is something "wrong" or "unusual" with the application, but may not be able to determine exactly what. But this may affect how the user perceives the application. If this hypothesis really is applicable on user interfaces as well, then it is maybe better to go with the Cordova+Ionic solution who has its own unique appearance.

**Empirical study on user interface and user experience**

The question if the Xamarin.Forms application is a victim of the Uncanny Valley effect or not will maybe be answered by the empirical study that was done on a small number of test persons. There were five persons who participated in the study, all with different background, each test person evaluated each prototype application by testing them on all platforms and was asked to assess them based on these four different aspects:

1. **Intuitiveness** - How easy it is understand and use the application without explicit instructions.
2. **Recognition factor** - To what degree, in terms of appearance, is the application similar to native applications.
3. **Look** - How attractive is the appearance of the user interface.
4. **Feel** - How does the application feel, in terms of smoothness and responsiveness.

The test persons was asked to grade the fulfilment of these factors from one (1) to five (5), where five is best/highest, and was also be asked to leave a comprehensive comment to each aspect. When testing the applications the test person will not know which application that
was developed using what tool. The result of this small empirical study is visualized in the table below.

Table 7: Result of the empirical study, presenting the combined score for an application on any platform.

<table>
<thead>
<tr>
<th></th>
<th>Cordova + Ionic</th>
<th>Native Android</th>
<th>Xamarin.Forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intuitiveness</td>
<td>3.5</td>
<td>4.5</td>
<td>4</td>
</tr>
<tr>
<td>Recognition factor</td>
<td>3.5</td>
<td>5</td>
<td>4.6</td>
</tr>
<tr>
<td>Look</td>
<td>3.5</td>
<td>5</td>
<td>4.7</td>
</tr>
<tr>
<td>Feel</td>
<td>3</td>
<td>4.7</td>
<td>4.0</td>
</tr>
<tr>
<td><strong>Avg sum</strong></td>
<td><strong>3.4</strong></td>
<td><strong>4.8</strong></td>
<td><strong>4.3</strong></td>
</tr>
</tbody>
</table>

The numbers in table 7 is the average grade for that application when tested on all platforms and from all test persons. The test persons were as mentioned also asked to leave a comprehensive comment to each aspect. Something that was mentioned by more than once when evaluating the Xamarin.Forms application was that the scrolling was smooth when compared against other solutions, the slide-in menu however was mentioned to be a little viscous but over all, according to the test persons, this solution have succeeded in achieving a native-look. When it comes to the hybrid application comments were that the interface felt flat and sometimes a bit slow and not as nice looking as the other two applications. The test persons had no difficulties in distinguish which application that was developed using what tool.

We can from table 7 and from the comprehensive comments come to the conclusion that the Xamarin.Forms application have succeeded in archiving a native look and feel and will therefore revive a grade of five in when we look at the criterion user interface and user experience. However the hybrid application is not very impressive when it comes to appearance and user experience but it is mediocre and will therefore score a grade of three.

Launch time

This criterion focuses on how long time it takes from that the user launch an application to until it have fully started. To measure the launch time on Android we made use the DDMS feature LogCat which allows us to view logs emitted by a device. From those log entries we were then able read out how long time it took for a device to launch an application, an example of such log entry:

```
I/ActivityManager: Displayed vitec.exlog.tf/.Act.LoginAct: +317ms
```

Here +317ms is how long time it took, in this case, to launch the login activity, which is the first activity in the native Android application.

For measuring launch time on Windows Phone, the Windows Phone SDK includes a tool called Windows Phone Application Analysis [28] which is fully integrated into Visual Studio and can be used to preform analysis of a Windows Phone application and gather and analyse data like battery and memory usage, amount of data uploaded/downloaded and the launch time of an application.
On iOS we found no easy way of measuring the launch time so we recorded the emulator screen and was from that able to determine the actual launch time. The launch times for the prototype applications when executed on different platforms is visualized in the graph below:

![Application Launch Time](image)

The launch times presented in figure 22 is the average launch time of five independent runs of each application on the following devices:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Device</th>
<th>OS version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>LG Nexus 5X</td>
<td>Android 6.0</td>
</tr>
<tr>
<td>iOS</td>
<td>Emulator iPhone 5s</td>
<td>iOS 9</td>
</tr>
<tr>
<td>Windows Phone</td>
<td>Nokia Lumia 920</td>
<td>Windows Phone 8.1</td>
</tr>
</tbody>
</table>

We can in the figure 22 over application launch time see that compared to native Android the two cross-platform solutions do not stand a chance, the native application launches almost two seconds faster. What is also notable is that on Windows Phone the Xamarin.Forms Application is more than twice as fast as the hybrid application, the reason of this is mainly because a Xamarin application is basically a native Windows Phone application since both of them is written using C#. In the combat between Xamarin and Cordova+Ionic Xamarin is the winner and will get a grade of three while Cordova+Ionic will be graded a two. Something that could have effected the result on iOS is that there were no actual hardware device was available for testing iOS applications, so we had to use an emulator.
Deployment

If you are building your application using the hybrid approach with Cordova + Ionic or any other similar tool you will be able to distribute your hybrid application to end users by publishing it to Google Play\(^1\), App Store\(^2\) or to the Windows Phone Store\(^3\). The same goes for an application built using the Xamarin platform, they can also be distributed through respective platforms application store, exactly like a native application. So for the end-user there is no differences in terms of how to install, download and update an application built using these approaches.

However there is a difference between the two tools Cordova + Ionic and Xamarin when it comes to with what ease you can publish an application to Apples App Store. You will in both cases need a Mac running OS X in order to publish an application for distribution via the Apple App Store. But when using Xamarin you can make use of the Xamarin Mac Agent\(^53\) to link your Windows machine to a Mac computer over SSH and then build, test and publish applications from your Windows machine and only use the Mac as a build server. This means that you do not need to move your entire codebase to a Mac computer in order to build an iOS application when using Xamarin, but this is something you will need to do if you are using Cordova + Ionic. Therefore Cordova+Ionic scores a grade of three, meaning that when it comes to deployment the Cordova + Ionic approach is fair but nothing more. Xamarin however gets a grade four, since it there is no need to switch to a Mac for building a iOS application.

\(^1\)https://play.google.com/store/apps
\(^2\)http://www.apple.com/itunes/charts/free-apps/
\(^3\)https://www.microsoft.com/en-us/store/apps/windows-phone
7 Evaluation Results

The result of our evaluation will be summarized and presented in table 8 below, we have graded the fulfilment of each criterion from part 2 of our evaluation framework from one (1) to five (5), where five is best/high/easy/most efficient.

Table 8: Result of the evaluation done on the two cross platform development tools.

<table>
<thead>
<tr>
<th>Evaluation Framework</th>
<th>Cordova + Ionic</th>
<th>Xamarin.Forms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Part 1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Properties</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developer</td>
<td>ASF + Ionic</td>
<td>Xamarin</td>
</tr>
<tr>
<td>First Released</td>
<td>2011 + 2013</td>
<td>2013</td>
</tr>
<tr>
<td>IDE</td>
<td>No official</td>
<td>Visual Studio &amp; Xamarin Studio</td>
</tr>
<tr>
<td>Licensing cost</td>
<td>Free</td>
<td>$999/year/developer</td>
</tr>
<tr>
<td>Device API access</td>
<td>Limited[14]</td>
<td>Full</td>
</tr>
<tr>
<td>Risk of vendor lock-in</td>
<td>None</td>
<td>Some</td>
</tr>
<tr>
<td><strong>Part 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Section 1, Development Perspective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation and support</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Maintainability</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Development speed</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Build-debug-test</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>CPU Usage</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td><strong>Section 2, User perspective</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Experience &amp; User Interface</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Launch Time</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Deployment</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>Sum part 2</strong></td>
<td>26</td>
<td>32</td>
</tr>
</tbody>
</table>

In the choice between the two cross platform development tools we have evaluated, we can see from the table above that Xamarin.Forms is probably the tool that will fit most people, since it has the highest score when we summarize the grades from part 2. But the result table should more be used as a look up table for developers when choosing what cross platform development tool to use.

Let's say that if you stand in front of the choice of what cross platform development tool to use and you know what decision factors that should be consider important when developing the mobile application, then we can use the result table to look up how well different tools fulfils that decision factor. If for example, development speed is more important than a highly polished UI in your application, then building a hybrid application using Cordova
+ Ionic is the most appropriate choice. But if we plan to build a computational heavy application then a tool that produce an application that uses memory and CPU power efficient is highly desirable, then Xamarin.Forms is a better choice then Cordova + Ionic.

7.1 Best cross platform development tool for Vitec

In discussions with Vitec, mentioned in Chapter 3 under the problem description, they described that the problems with their current application as they see it, is that it lacks completely the native feeling they wish to archive in terms of user experience. Their application also feels slow when switching between views and navigating through experience and the current application is also, according to the developers, hard to maintain. We asked one of the developers of the current “Teknisk Förvaltning” application to rank each criterion from part two of our evaluation framework by how important they are. The result can be seen in table 9 below. The ranking was performed by ranking each criterion between ten (10) and one (1) were a ten means that the criterion is considered to be the most important criterion.

Table 9: Ranked importance of criteria for Vitec’s application.

<table>
<thead>
<tr>
<th>Part 2</th>
<th>Ranked Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1, Development Perspective</td>
<td></td>
</tr>
<tr>
<td>Documentation and support</td>
<td>7</td>
</tr>
<tr>
<td>Maintainability</td>
<td>8</td>
</tr>
<tr>
<td>Development speed</td>
<td>4</td>
</tr>
<tr>
<td>Build-debug-test</td>
<td>6</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>2</td>
</tr>
<tr>
<td>CPU Usage</td>
<td>2</td>
</tr>
<tr>
<td>Section 2, User perspective</td>
<td></td>
</tr>
<tr>
<td>User Experience &amp; User Interface</td>
<td>10</td>
</tr>
<tr>
<td>Launch Time</td>
<td>9</td>
</tr>
<tr>
<td>Deployment</td>
<td>7</td>
</tr>
</tbody>
</table>

As we can see the user experience and the user interface is ranked by Vitec to be the most important decision factors, performance in terms of launch time is ranked second and maintainability is ranked third. Least important is efficient use of memory and CPU power. This is expected since their application is not very computational- or graphically heavy and therefore is this criterion not of great importance. If we look up these decision factor that Vitec does consider to be important against the result table 8 we come to the conclusion that Xamarin.Forms is the tools that most likely will fit their needs best. Since that tool produces an application with a good and smooth, almost native, user interface and experience and it is developed using C# which will be easier to maintain than a hybrid application built using JavaScript.
8 Conclusions

To evaluate something objectively is very difficult, it is easy that it boil down to subjective conclusions. Which in itself is not entirely uninteresting, but subjective conclusions tend to be the pretty individual. Either way, I think we have managed to achieve what we have described in the problem specification:

- Can we develop a generic evaluation framework that can be used to assess any cross-platform mobile development tool, with the aim to help developers select the most appropriate tool for a specific mobile application.

We have developed an evaluation framework that can be used to assess any cross-platform development tool. We were also able to, by using the developed evaluation framework and by ranking the importance of each evaluation criterion, suggest which tool of those we have evaluated in this thesis who is the best fit for Vitec and the development of their mobile application "Teknisk Förvaltning".

8.1 Future Work

As of now only two cross-platform development tools have been evaluated using the evaluation framework, for the result table seen in table 8 to be more extensive more tools will need to be evaluated using the framework.

It would also be interesting to see different types of applications being evaluated using the framework. Vitec’s application is very much a client-server application. But it would be interesting to see if there is a difference in the end result, if the prototype application is of another type, say a more computational heavy application or even a game. This would be interesting to investigate further.

When it comes to further refinement of the evaluation framework the two criteria Memory- and CPU usage under section 1, part 2 of the evaluation framework. Were we analyse how efficient the different implementations use memory and CPU power is now only evaluated on the Android platform. This should be extended by doing the same type of analysis on both the iOS and the Windows Phone platform.
Acknowledgements

First of all I would like to thank Vitec for giving me the opportunity to do this thesis. I would also like to thank my supervisor Pär Ådhén at Vitec, who gave me valuable feedback throughout this thesis. Thanks to my supervisor at Umeå University, Anders Broberg for useful ideas, support and constructive criticism during the Master’s thesis work. And Last, but certainly not least, I would like to thank my future wife Maria, for your support.
References


