Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Sanna Ottka

Comparison of mobile application development tools for multi-platform industrial applications

Master's Thesis
Espoo, March 26, 2015

Supervisor: Professor Petri Vuorimaa
Advisors: Zhongliang Hu M.Sc. (Tech.)
Pasi Koivumäki M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Sanna Ottka

Title:
Comparison of mobile application development tools for multi-platform industrial applications

Date: March 26, 2015 **Pages:** vii + 68

Major: Media Technology **Code:** IL-3011

Supervisor: Professor Petri Vuorimaa

Advisors: Zhongliang Hu M.Sc. (Tech.)
Pasi Koivumäki M.Sc. (Tech.)

The mobile device base is strongly divided between different mobile platforms, most importantly Android, iOS and Windows Phone. Applications developed for one platform with traditional development methods only work on that platform, and supporting multiple platforms requires developing the application separately for each of the platforms. Different cross-platform methods have been introduced as a solution to this problem. They allow deploying the application for multiple platforms from a single code base.

This thesis studies when a cross-platform solution is a viable alternative to developing the application separately for each target platform. The focus is on industrial applications that communicate wirelessly with electric drives. Three main development methods are identified as native, web and hybrid applications. Native applications are developed for a single platform with the platform's standard development tools and conventions. Web applications are developed with web technologies and hosted as web sites. Hybrid applications utilize web technologies but are installed on the device like native applications and are developed with specialized cross-platform tools such as PhoneGap.

Each method is evaluated by various criteria. The evaluations show that on average the methods are almost equal, and their suitability depends on the target application. A matrix is generated that calculates the best fitting method for a given application with the help of application-specific weight values. The weight values represent how important each individual criterion is for that application. The matrix is then used to select the best method for three study cases. The results show that a cross-platform solution can sometimes be a viable option, based on the needs of the application.

Keywords: mobile, cross-platform, native, web, hybrid, industry

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Sanna Ottka		
Työn nimi:	Vertailu mobiilisovelluskehityksen työkaluista teollisuuskäyttöön tarkoitetuissa monialustaisissa sovelluksissa		
Päiväys:	26. maaliskuuta 2015	**Sivumäärä:**	vii + 68
Pääaine:	Mediatekniikka	**Koodi:**	IL-3011
Valvoja:	Professori Petri Vuorimaa		
Ohjaajat:	Diplomi-insinööri Zhongliang Hu Diplomi-insinööri Pasi Koivumäki		

Mobiililaitekanta on voimakkaasti jakautunut eri mobiilialustojen välille, joista tärkeimmät ovat Android, iOS ja Windows Phone. Yhdelle alustalle perinteisillä kehitysmenetelmillä kehitetyt sovellukset toimivat vain sillä alustalla, ja usean alustan tukeminen vaatii sovelluksen kehitystä jokaiselle alustalle erikseen. Ratkaisuksi tähän ongelmaan on esitetty erilaisia alustariippumattomia menetelmiä, jotka mahdollistavat sovelluksen julkaisemisen useammalle alustalle yhteisestä lähdekoodista.

Tässä työssä tutkitaan milloin alustariippumaton ratkaisu on varteenotettava vaihtoehto sovelluksen kehitykselle erikseen jokaiselle alustalle. Päähuomio on teollisuussovelluksilla, jotka kommunikoivat langattomasti taajuusmuuttajien kanssa. Kolme tarkasteltavaa menetelmää ovat natiivi-, web- ja hybridisovellus. Natiivisovelluksia kehitetään yhdelle alustalle alustan tyypillisillä kehitystyökaluilla ja menetelmillä. Websovelluksia kehitetään webtekniikoilla ja tarjotaan nettisivuina. Hybridisovellukset käyttävät webtekniikoita mutta ne asennetaan laitteeseen natiivisovellusten tavoin. Niitä kehitetään erikoistuneilla alustariippumattomilla työkaluilla kuten PhoneGap:llä.

Jokainen menetelmä arvostellaan erilaisten kriteerien perusteella. Arvostelut osoittavat että keskimäärin menetelmät ovat lähes yhtä hyviä, ja niiden soveltuvuus riippuu kohdesovelluksesta. Tähän kehitetään matriisi joka valitsee parhaan menetelmän kohdesovellukselle sovelluskohtaisten painokertoimien avulla. Painokertoimet kuvaavat kuinka tärkeä tietty kriteeri on sovellukselle. Matriisia käyttämällä valitaan parhaat kehitysmenetelmät kolmelle esimerkkisovellukselle. Tulokset osoittavat että alustariippumaton ratkaisu voi tietyissä tapauksissa olla mahdollinen, riippuen sovelluksen tarpeista.

Asiasanat:	mobiili, alustariippumaton, natiivi, web, hybridi, teollisuus
Kieli:	Englanti

Acknowledgements

Thank you Zhongliang (Ross) and Pasi for your invaluable advice and support. Thank you Petri for keeping me on the right track. Thanks to Mikko, Martti, Teemu, Kimmo, Lauri and Steffen for sharing your expertise. Thank you Helge and Antti for your support. And thank you Ossi for reminding me that there's life after thesis.

Espoo, March 26, 2015

Sanna Ottka

Abbreviations and Acronyms

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
URL	Uniform Resource Locator
API	Application Programming Interface
SDK	Software Development Kit
UI	User Interface
GUI	Graphical User Interface
IDE	Integrated Development Environment

Contents

Chapter 1

Introduction

Today mobile devices are everywhere. Nearly everyone has a smartphone, and often a tablet as well. Over 1,8 billion mobile phones and 270 million tablets were shipped in 2014.[25] By the end of year 2014 over 5,6 billion people worldwide had at least one mobile device, with an average of 1,36 devices per user.[39] Mobile devices offer new possibilities for service providers, as users can access their services anytime, anywhere, and carry the entire internet in their pockets. Mobile applications can replace traditional tools like calendars, contact lists or maps in a compact and portable form. They also bring new tools to the consumers, such as activity trackers, navigation services or communications with other devices. In the industrial setting, a user could monitor and control the machinery around them right at the factory floor, with their handheld device communicating wirelessly with the machinery. There is no need to set up a stationary desktop system or carry around a heavy laptop, when most tasks can simply be carried out with a smartphone. Therefore, it becomes increasingly useful to offer the services and functionalities as mobile applications, accessible with smartphones or tablets. But, offering services as mobile applications comes with its own challenges.

The mobile environment is highly fragmented. There are different mobile platforms, such as Android, iOS, Windows Phone and BlackBerry, and each has multiple versions in active use. The devices themselves are physically different depending on the manufacturer, and the device vendors often offer their own customized versions of the standard operating systems. In such a world offering an application for just a single mobile platform would severely limit the potential userbase, thus limiting the usefulness of the application. Each platform has its own programming language, device interface, development tools and style conventions. An application developed for one platform with the platform's own development tools will not work on other platforms. But developing a dedicated application separately for each individual plat-

form is a time-consuming and expensive undertaking. There needs to be a better solution for handling an application intended for multiple platforms.

There are two main solutions for this problem. The first option is to offer the service over the internet, as a web application. Web application is a specific type of web site that is dedicated for mobile devices. The application is accessed through the device's browser like any other web site, and the same application works for all platforms. The second option is to use a cross-platform tool. These tools can create the application for different platforms from a common code base. They operate under the philosophy of write once, run anywhere. There are several cross-platform tools available, as both commercial and open-source versions.

But, are these cross-platform solutions a viable alternative to developing a dedicated application separately for each platform? The aim of this thesis is to compare the methods to each other and find the factors that determine which method should be chosen. As different applications have different needs and it is unlikely that one solution will suit everything, three applications are included as study cases. The thesis introduces a methodology for choosing the most suitable development method for a given application, based on the comparison of methods and the needs of the application. The methodology is tested on the three example applications to validate it.

Figure 1.1: ABB's low voltage drive ACS580-01.

1.1 Background

The thesis studies mobile application development in ABB, a multinational industry corporation, operating in fields such as robotics, power and automation technologies. ABB employs over 140 000 people in a hundred countries, and roughly 5200 in Finland.[1] ABB is expanding its service range to mobile platforms and this thesis supports the process. The context of the thesis is industrial mobile applications, but the majority of the results apply to other types of mobile applications as well.

The main focus is on applications that operate with electric drives, such as the low voltage drive ACS580-01 shown in figure 1.1. A drive is a system connected to an electric motor that controls how much energy the motor receives. An electric motor's rotating speed depends on the amount of energy it uses. Using a drive makes it possible to match the motor's speed to the current demand and therefore save energy and minimize waste.[15]

1.2 Problem Statement and Research Questions

This thesis has two main purposes. The first purpose is to provide an overview of the currently relevant technologies in the mobile environment to support businesses looking to expand their services to mobile devices. This overview contains the current major mobile platforms, types of mobile applications and mobile application development methods and tools. The second purpose is to produce a comparison of the currently available methods for developing mobile applications targeting multiple platforms and providing guidelines for choosing the most suitable development method for a given use case. Only applications that need to support more than one platform are included in the scope of this thesis. The comparison is used for selecting the best fitting tools for three different use cases in ABB. The following research questions will aid in this task:

- What are the key differences between the mobile application development methods?
- How do the required technical features limit the choice of mobile application development tool?
- Is a cross-platform solution a viable alternative to developing separate applications for each platform?

1.3 Structure of the Thesis

The second chapter gives a brief overview of the methodology used in this thesis. The third chapter examines the different mobile platforms and their current standings in the market, as well as the different types of mobile applications and their development methods. The fourth chapter contains a summary of the available cross-platform tools that are likely to be suitable for ABB's use cases. The fifth chapter determines the criteria to be used for comparing the development methods. The sixth chapter contains the evaluations for each of the different methods for creating mobile applications based on the criteria presented in fifth chapter. The summary of the evaluations is collected in a results matrix for later use. The seventh chapter expands the results matrix into a tool selection matrix and uses it to select the best tool for three example applications. The findings and the future are discussed in the eighth chapter, and finally the ninth chapter draws the conclusions of the thesis.

Chapter 2

Methodology

2.1 Literature Review

Background information on mobile platforms and applications is researched through a literature review. Online search engines and databases, including Google Scholar and IEEE Xplore, provide general information on the subject with keywords such as "mobile applications", "mobile application development" and "cross platform mobile application development". More information on individual platforms is gained by searching with the platform's name and visiting the developer's sites. For application types and their differences keywords "native applications", "web applications", "hybrid applications" and "native vs. web comparison" are used.

Up to three pages of results for each search are included in the following process:

- Only include papers that are available in digital format through Aalto University library collections, and are written in English or Finnish.
- Read the title and abstract of the paper. Discard the paper if it is not relevant to the topic.
- Glance over the paper to get an overview of the contents and quality. Discard the paper if it is not of sufficient quality or it is not relevant to the topic.
- Read the remaining papers in full.

In addition, any relevant references in the articles are examined for additional material.

2.2 Searching and Choosing Cross-Platform Tools for Analysis

During the literature review, the cross-platform tools mentioned in the read articles are collected in an initial list. The next step is to perform a search in Google using keywords "cross platform mobile development tools" and check the first two pages of results for any additional tools and add them to the list. Each tool is then briefly reviewed by visiting the developer's web site and doing a general web search with the tool's name.

The tool must support at least the two most popular mobile platforms: Android and iOS, to be considered a viable choice. Additionally, as many of the articles are few years old, the tools mentioned in them may not be maintained anymore. Therefore, the tool also needs to be in an actively developed state. Finally, one of the main uses cases examined in this thesis includes communication between the drive and mobile device over Bluetooth. Therefore, another requirement is that the tool has some way to access Bluetooth functionality, either as a standard feature or a plugin, or offers the possibility to customize the tool or write platform specific native code. Any tools that do not fulfill these three criteria are disregarded. Then, the list of tools is narrowed down to a maximum of five most promising tools. This is done based on the tool's popularity among developers, amount of supported platforms and hardware features, and the maturity of the tool. The selected tools should also represent different technological approaches. The remaining five tools are discussed in this thesis.

2.3 The Evaluation and Comparison Methods

The criteria for comparing the tools are gathered from multiple sources. The initial list is gathered from criteria encountered during the literature review, and then modified and extended for ABB's use case. This is done by taking into account ABB's guidelines for developing mobile applications and the results of an external user study performed by Idean for ABB's applications. The list of criteria is then presented to mobile development experts in a series of interviews and further refined based on their feedback. The interviews are performed with the active interview methodology with semistandardized structure.[14] The interviews follow a set schedule, where the subject is first presented with the list of criteria and asked to give their opinion on each. Then they are asked the following set of questions:

- Which platforms do you think are important to support and why?
- What are the most important differences between the different development methods?
- How would you decide whether to develop an application as a native, web or hybrid application?
- Have you used cross-platform tools, and if so, which one and how was your experience?
- How would you expect the mobile application development to evolve in the next five years?

The semistandardized interview also allows unscheduled questions and probes during the interview if any new subjects emerge that are not sufficiently covered by the standard questions. The interviews are recorded and their results are used to validate and modify the list of criteria.

The three development methods: native, web and hybrid are then evaluated against the criteria. One cross-platform tool will represent the hybrid application, as the results for different tools would be similar. A verbal description examines how well the method performs on that criterion, and a numeric value from 1 (very poor) to 5 (very good) is given for comparing the methods. The first set of evaluations are based on the author's knowledge of the subject, other evaluations in literature and the results of the interviews. In addition, a web questionnaire is used to gather another set of evaluations by gathering the opinions of mobile application developers. The survey is produced with Google Forms and distributed via a web link. The survey will ask the subject to rate each of the methods by each of the criteria from 1 (very poor) to 5 (very good). If the subject has no opinion on some criteria or method they can leave the field empty. Screenshots of the survey are included in Appendices A.1 to A.4 The survey is distributed to 5 LinkedIn discussion groups, each with several thousand members, and to developers of ABB subcontractor company. The survey received 10 replies. The values gained from the survey are compared to the previous evaluations and used to validate the evaluations. Also, based on the results it is decided if the survey is a suitable method for collecting data for the evaluation.

The evaluation results will be processed in an Excel table, as illustrated in figure 2.1. The evaluation method used in this thesis is similar to the Decision Matrix method[12], with application-specific multipliers for each criteria. The numeric evaluations for each of the tools are gathered in a table, forming an evaluation matrix. Below the matrix are two results vectors.

Criteria	Development tool Method 1	⋮	Method m	Weight values Application
Criteria 1	x_{11}	...	x_{1m}	y_1
Criteria 2	x_{21}	...	x_{2m}	y_2
...	...	...	...	...
Criteria n	x_{n1}	...	x_{nm}	y_n
total	SUM(x_{11},...,x_{n1})	...	SUM(x_{1m},...,x_{nm})	
average	AVERAGE(x_{11},...,x_{n1})	...	AVERAGE(x_{1m},...,x_{nm})	
Weighted averages Application	SUM(x_{11}*y_1,...,x_{n1}*y_n) /SUM(y_1,...,y_n)	...	SUM(x_{1m}*y_1,...,x_{nm}*y_n) /SUM(y_1,...,y_n)	

Figure 2.1: The tool selection matrix formula.

The first vector contains the sums of the evaluations for each method. The second vector shows the averages of the results for each method. Adding weight multipliers for the criteria will form a tool selection matrix for choosing the best approach for a specific use case. The weight values describe how important the criterion is for the use case and are determined as 1 (not very important) to 3 (very important). Individual criteria can also be excluded from the comparison if they do not apply for the use case by setting their weights to zero. The weight values for a single application are collected in a weight vector and shown on the right side of the matrix. The final result vector then shows the weighted average for each method for that application. The weighted results represent how well the method fits for the application, and the method with the highest average is the best fit for the application. The entire table with the evaluation matrix and weight and result vectors forms the tool selection matrix.

The selection matrix is validated with two existing ABB applications by comparing the approach that was used for creating the application to the one suggested by the matrix. Then the matrix is used to recommend a tool to use in creating a third application in a future project in ABB. For the three

example applications the weight values are gathered by interviewing their product managers. The interviews follow a standardized structure.[14] The interviewee is asked to go through the list of criteria and tell how important each criterion is for the application on a scale of 1 to 3 and why. The ratings gained from the interview are used as weight values in the tool selection matrix. If the answers from different people differ, their averages are used. Short summaries of the reasons for the weight values are presented in the criteria weight lists.

Before using the selection matrix to choose the best method, the methods are prescreened to remove impossible solutions. In the context of this thesis, the only case where this is used is when a certain native functionality is needed, but is not supported by some methods. The prescreening does not affect the calculations, as the matrix calculates the results automatically for each method with no extra work from the user. It is therefore enough to acknowledge that a certain method is not a possible solution, regardless of the rating given by the matrix.

Chapter 3

Mobile Application Development

3.1 Mobile Platforms

According to a market share study by IDC [29], the smartphone market is currently clearly dominated by Android, which held over 84 percent of the market during the second quarter of 2014. Meanwhile, iOS still remains strong with 11,7 percent of the market share, although it has lost 6,5 percentage points over the last three years. Windows Phone has grown steadily, and currently holds 2,5 percent of the market, while BlackBerry OS has plummeted from 12,6 percent in 2011 to a mere half a percent of the market share today. This leaves just 0,7 percent of the market to other platforms, such as Firefox OS by Mozilla and Sailfish OS by Jolla. The market shares of the top four mobile platforms and the remaining market during the second quarter of each year between 2011 and 2014 are shown in table 3.1.

On the tablet side, the market is more evenly divided between the two giants, Android and iOS. A tablet sales study by Gartner [24] shows that Android and iOS were almost even in 2012 with market shares of 45,8 and 52,8 percent, respectively. In 2013 Android claimed a clear lead with 61,9 percent of the market, and iOS's share has decreased to 36 percent. Microsoft

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
Q2 2014	84,7%	11,7%	2,5%	0,5%	0,7%
Q2 2013	79,6%	13,0%	3,4%	2,8%	1,2%
Q2 2012	69,3%	16,6%	3,1%	4,9%	6,1%
Q2 2011	36,1%	18,3%	1,2%	13,6%	30,8%

Table 3.1: Smartphone Operating System market shares from years 2011 to 2014.[29]

Operating System	2013 Market Share (%)	2012 Market Share (%)
Android	61,9	45,8
iOS	36,0	52,8
Microsoft	2,1	1,0
Others	<0,1	0,3

Table 3.2: Tablet Operating System market shares in 2012 and 2013.[24]

held 2,1 percent of the market in 2013 with its Windows 8 system. Other platforms are nearly nonexistent in the tablet market, holding less than 0,1 percent of the market. The market shares for tablet operating systems are also displayed in table 3.2.

The next section will take a closer look at the three most popular mobile operating systems, Android, iOS and Windows Phone.

3.1.1 Android

Android is a full set of software for mobile devices and includes an operating system, middleware and key mobile applications. Android was originally created by Android Inc., which was bought by Google in 2005.[21] Google published Android in 2007 and formed the Open Handset Alliance (OHA) with a group of other technology companies, device manufacturers and wireless carriers.[4] Today Open Handset Alliance houses more than 80 companies, dedicated to developing and distributing Android.[2]

Google offers Android as an open source solution, with anyone being able to download the source and modify it. Several handset vendors, such as Samsung, HTC and LG, offer their mobile devices with the Android OS, usually as their own customized version.[17] While this offers more variety for the user looking for a mobile device, it has caused the Android version landscape to become fragmented. The vendors need time to update their own versions of Android, and therefore the updates often become available for the users later than the official releases. The vendors may also skip some smaller version changes, and only update their versions for major releases. In addition, the devices use different hardware, for example, different sized screens and physical buttons. Application developers need to take this into account, and ensure their applications are compatible with older platform versions or some users may be unable to use them.[31]

Android is designed as a stack of various components, divided into five layers. Figure 3.1 illustrates the different layers of the system. Android is built on a modified Linux 2.6 series kernel that provides core system services such as security, memory management, process management, network

Figure 3.1: Android system architecture.[43]

stack and driver model. The kernel and low level tools are contained in the bottom layer, colored red in the illustration. The basic libraries included in Android are programmed in C and C++, and are accessed through the Android application framework.[43]

The Android runtime contains a set of Java core libraries and the Dalvik virtual machine (VM). The Dalvik VM executes files in Dalvik Executable (.dex) format, usually transformed from Java byte code to Dalvik byte code.[16] Every Android application runs in its own process with its own sandboxed instance of Dalvik VM. Dalvik has been optimized so that a device can run multiple VMs at the same time efficiently.[43] The kernel also provides an abstraction of the underlying hardware for the rest of the software stack.[23]

The application framework layer gives the developers access to the same framework Application Programming Interfaces (API) used by the core applications. [43] The frameworks are written in Java and provide abstractions of the Android libraries and the features of the Dalvik VM.[16]

On top of the stack reside the applications. Both core and third-party applications have equal access to the device's capabilities, such as making

calls or using the camera. They can also access capabilities of other applications, within security constraints. Users can also replace any core application with another application, and have their device use the new application by default.[3]

Applications for Android are developed through the Android Software Development Kit (SDK), usually with the Java programming language. The SDK provides the API libraries and developer tools for building, testing and debugging for Android. Development can be done in any of the current major operating systems and an integrated development environment (IDE) of choice, although Google recommends using Android Studio. Another common option is to use Eclipse IDE with Android Developer Tools (ADT) plug-in, provided by Google, which integrates the Android SDK into Eclipse. The ADT allows the developer to test the application with an Android emulator or a connected device, and provides a graphical editor for building the user interface (UI) of the application.[5]

The main distribution channel for Android applications is the Google Play Store, formerly known as Android Market, where developers can publish their applications after registration. Unlike the process used by Apple, Google does not require the application to be pre-approved to appear in the Play Store.[23] Android applications can also be freely distributed through other channels as well, such as the developer's own web page or third-party application stores.[6]

3.1.2 iOS

Apple's iOS was originally released in 2007 along with the first iPhone, and known as iPhone OS. Since then it has been extended to support other Apple devices as well, such as iPad and iPod and renamed iOS. Unlike Android, iOS has a relatively small number of versions, all provided by Apple. It also has only a small number of different devices, also only supported by Apple. This severely reduces the number of versions of the OS and hardware application developers need to support.[47]

The iOS system architecture is divided into four layers, illustrated in figure 3.2. The layers provide different levels of abstraction between the applications and the underlying hardware. The various core frameworks are written in the Objective-C programming language. The Core OS layer contains the low-level features most of the other technologies are built upon. Applications rarely use these technologies directly, but rather use them through the other frameworks. However, the layer contains frameworks for features such as security, Bluetooth support and communicating with external hardware, that can be used by applications if needed. The layer also contains the kernel

environment, drivers and low-level UNIX interfaces of the operating system. Access to the kernel and drivers is restricted to a limited set of system frameworks and applications.[9]

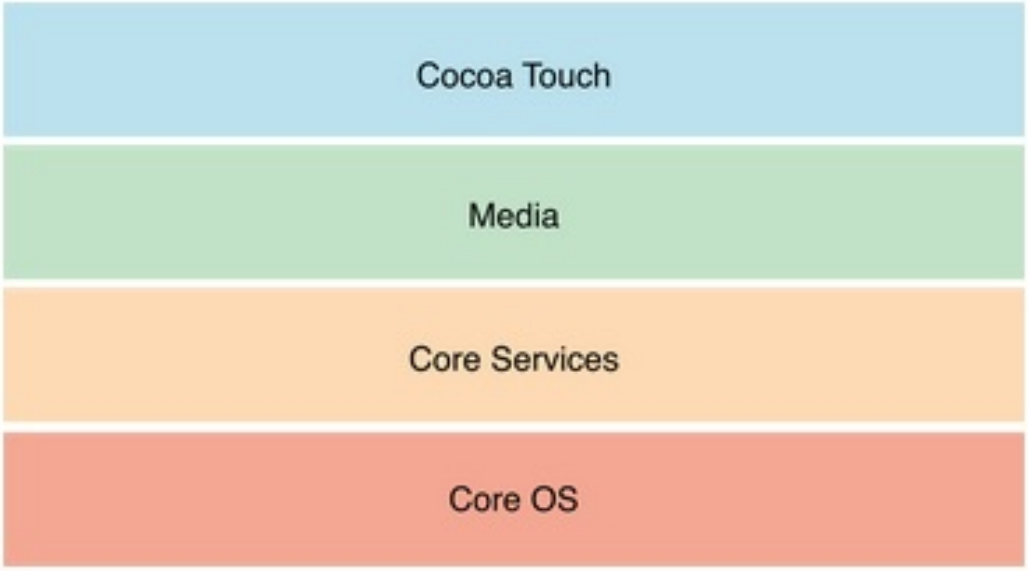


Figure 3.2: iOS system architecture.[9]

The various system services used by applications are contained in the Core Services layer. This includes technologies to support features like location, iCloud storage, peer-to-peer services and networking. The Media layer above it contains the graphics, audio and video technologies needed to implement multimedia features in applications. Finally in the top layer resides the Cocoa Touch framework, providing the key frameworks for building iOS applications. This includes high-level programming interfaces for making animations, networking and modifying the appearance of the application. Cocoa Touch also handles touch-based inputs and multitasking.[8]

Building iOS applications requires using Apple's Xcode IDE on a Mac computer running OS X 10.8 or later and iOS SDK. Xcode provides the standard tools to code, debug and design the interface for the applications. Generally iOS applications are written in Objective-C language.[8]

Applications for iOS are distributed to consumers exclusively through Apple's App Store. Developers enroll in Apple Developer Program and pay a yearly fee to be able to publish applications in the App Store, and applications go through an approval process by Apple before appearing in the store.[8] The approval process causes longer development times, but lowers

the number of low-quality applications in the store.[17] The approval process can pose a challenge for applications developed with various cross-platform methods. For example, in 2010 Apple maintained that apps must be "originally written in Objective-C, C, C++ or JavaScript" to be accepted into the store. The restrictions have been eased since then, but applications still sometimes get rejected for being too slow or not feeling native enough. Apple App Store can also reject apps that download executable code or interpret code not contained within the application archive.[32]

3.1.3 Windows Phone

Windows Phone is a mobile operating system developed by Microsoft. The first version, Windows Phone 7 was released in 2010 as a successor to Windows Mobile. It was based on Windows CE with the user interface redesigned, and was targeted more towards the consumer market unlike its predecessor, which was designed for corporate use. Windows Phone is carried by several different mobile device vendors, such as Nokia, HTC and Samsung. Windows Phone 7 had severe limitations in place regarding hardware, causing major challenges for the device vendors especially towards the end of its life-cycle. The update to Windows Phone 8 removed or reduced most of the limitations and, for example, added support for multi-core processors and higher screen resolutions.[46]

Figure 3.3 provides a high-level overview of the architecture of Windows Phone. The bottom level contains the drivers required to support the phone hardware, supplied by the device vendor. The second level contains the kernel. Windows Phone 8 switched to Windows NT kernel, which is also used by Windows 8 for tabletop computers, instead of the Windows CE used by Windows Phone 7. The operating system and kernel largely come from Windows with Windows Phone specific modifications. This simplifies the process of porting applications between tabletop and mobile devices. Windows Phone 8 uses Core System from Windows, which handles system features such as booting, managing hardware and resources and communications with networks. The Core System is supplemented by the Mobile Core, a set of Windows Phone specific binaries for handling phone-specific tasks. Above the kernel are the shared services of the system and APIs used by applications to access the underlying functionality. The layer contains many of the same features used by Windows 8, such as the NTFS file system, DirectX graphics engine, networking stack and security elements.[53]

Developing applications for Windows Phone 8 requires a computer with 64-bit Windows 8. However, applications for the most recent update, Windows Phone 8.1, can be developed with 32-bit Windows 7 or higher. Devel-

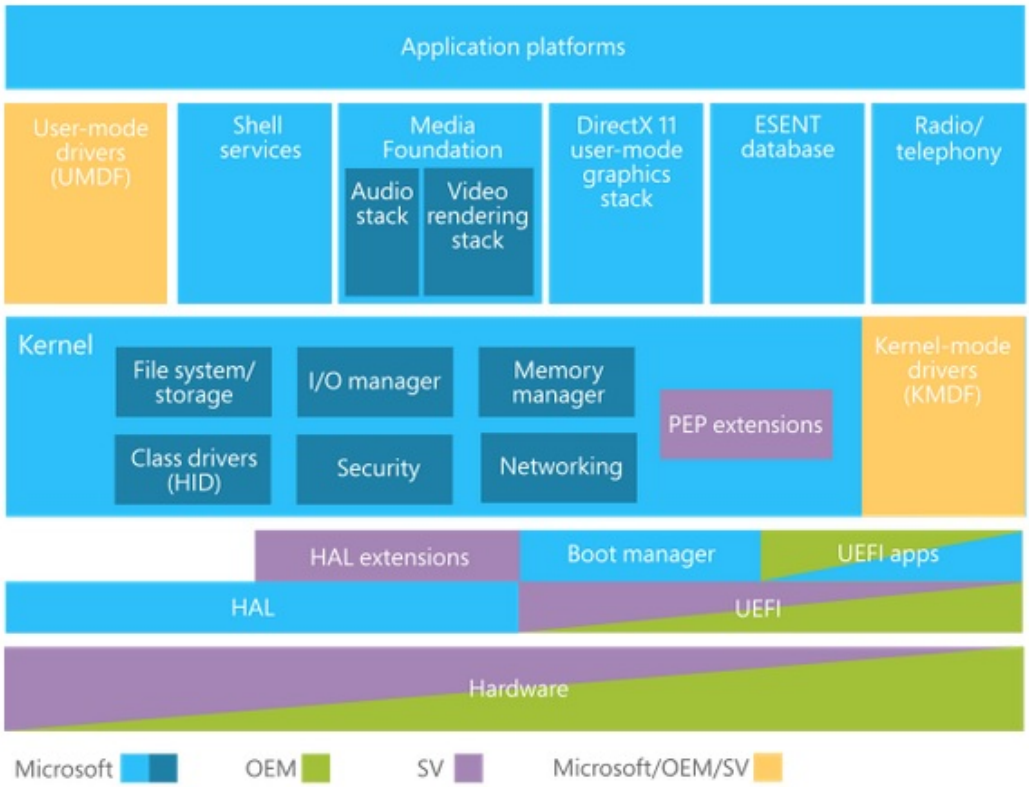


Figure 3.3: Windows Phone system architecture.[53]

opment is done through the Windows Phone SDK and Driver Kit offered by Microsoft. The SDK includes all the tools needed for development, including Microsoft Visual Studio Express for Windows Phone and Windows Phone emulator. The SDK can also be integrated into Visual Studio Professional or higher. Applications can be programmed in C#, Visual Basic.NET or C++.[53]

Similar to Apple, Windows Phone applications can only be distributed through the Windows Phone Store, and they must first go through an approval process performed by Microsoft.[53]

3.2 Mobile Applications

Mobile applications can be divided into two groups: integrated and nonintegrated applications. Integrated applications consist of a package that is installed on the device. The application is integrated into the system when installed, it appears in the default application list and can access system functionalities. Integrated applications include native applications that use platform-specific technologies and libraries, and applications that combine web and native technologies, known as hybrid applications. Nonintegrated applications on the other hand do not run directly on the device. Instead they are invoked through another application, generally a web browser. Applications that run in the web browser are known as web applications.[32]

3.2.1 Native Applications

Native applications are developed for a specific target platform, using that platform's SDK and frameworks, and the app is tied to that specific environment. For example, an Android application is developed with Java using the Android SDK and the APIs provided by Android and uses platform-provided elements for rendering the UI.[5] If the developer wants to support multiple platforms with pure native applications, the applications need to be developed separately for each platform. Native applications are installed on the device from the platform's app store or by using other platform-specific installation methods. They have access to all the native features offered by the device.[27]

Individual platform's SDKs usually provide graphical editors for building user interfaces, and have efficient abstractions for common UI controls and experiences. Each platform also has its own guidelines and best practices for building UIs, so that different applications for the same platform have the same visual style and feel. Common UI elements and conventions also vary from platform to platform. For example, Android devices have some form of a back button by default, whereas iOS applications usually have a button in the application itself with the same functionality. Native application built for a specific platform can easily follow these guidelines and provides the best user experience expected for that platform.[18]

The developer can also use a cross-platform development tool to create native applications for multiple platforms from the same code base. Applications developed this way are very similar to native applications, but cannot be considered true native as they have some key differences to applications developed using the platform-appropriate SDK and development methods.

The Other Methods section will cover such cases.

3.2.2 Web Applications

Mobile Web applications take advantage of the standardized Web technologies and good browser support of mobile platforms. The application is implemented as a single Web site optimized for mobile devices. The site can be accessed on any mobile device, and behaves largely the same way regardless of platform. The application is not installed on the device, but instead it is retrieved via a Uniform Resource Locator (URL) using a mobile browser on the device, and most of the functionality is stored on the web server. This makes maintenance of the application simple, as an update only requires the user to reload the page. It does, however, require constant network connection in most cases, and disconnections may cause serious issues in some applications.[27]

Developing a mobile web application is very similar to developing a web site. It uses the same technologies and languages with mobile-specific modifications such as accessing the platform API and adapting to different screen sizes. Web applications often follow the standard three-tier architecture, where the application is divided into presentation, logic, and data tiers. Client side of the applications includes the presentation tier, and server side includes the logic and data tiers. Each tier can have its own selection of programming languages and concepts and therefore require a different set of skills. Recently new unified solutions have also been introduced, that allow developing all three tiers with a single tier's technologies.[30]

Web applications by default have the look and feel of a web site, and are mostly consistent across different platforms. Different browsers can still have small differences in how they render the sites, and some platform-specific programming is usually required due to differences in hardware and platform conventions, especially when targeting both tablet and smartphone markets.[18] The Graphical User Interface (GUI) can also be made to imitate native GUIs by using OS-specific Cascading Style Sheets (CSS) files if wanted. This must be done separately for each platform, and getting the GUI to be accurate enough imitation of a native GUI can be challenging.[32]

The main disadvantage of web applications is their limited access to device functionality. Since they run in the web browser they are in a heavily restricted sandbox, and do not have direct access to lower-level APIs operator systems offer to native applications. These APIs include, for example, access to storage, device sensors, camera, and Bluetooth connection.[18] The latest version of HyperText Markup Language (HTML), HTML5 seeks to solve many of these problems. It has been gaining popularity over several years

now, but the browser support is still somewhat lacking especially on desktop browsers. However, nearly every mobile browser supports most HTML5 features by now.[28] It adds many new features over its predecessor, HTML4, including APIs that provide access to some device functionalities that were inaccessible to web applications before. For instance, web applications using HTML5 can access the device camera and save data for offline use.[48]

3.2.3 Hybrid Applications

Hybrid applications are an attempt to resolve the issue with web applications' limited access to hardware, while still employing common Web technologies, by combining web and native technologies. Generally, the application is implemented as a web page using standard HTML technologies, but instead of displaying it in a web browser, it is wrapped as a native application using a specialized engine. The page is displayed in a regular WebView or similar feature inside the application, and the engine provides frameworks for accessing the device's native functionalities, usually through JavaScript.[27]

Hybrid applications are installed on the device the same way as native applications. Hybrid application caches its data locally on the device on installation, removing the need for constant data connection. This also increases its speed and responsiveness over a pure web application.[26] Like web applications, hybrid applications can use web technologies to render a web-like GUI or imitate native GUIs. Alternatively, they can use native methods for rendering a native GUI if the engine provides support for it. Different engines provide different levels of abstraction for building a native GUI, and determine how much of the GUI code base can be common for all platforms and how much must be written separately for each platform.[32]

3.2.4 Other methods

There are still other methods of creating mobile applications that do not directly fall into any of the previously examined categories. These are applications that run on the device like native applications, but have been created from a common code base for multiple platforms using a cross-platform development tool. These tools provide varying layers of abstraction between the application code and native functionality, allowing the developer to separate the common business logic from device-specific code.[52]

Ohrt and Turau [32] identify the following four types of applications created with cross-platform tools:

- Purely native applications that access the system API directly

- Purely native applications that access the system through an abstraction provided by the library
- Interpreted applications that include the VM in the application package
- Interpreted applications that require the VM to be installed as a separate application.

A common way to create native applications for multiple platforms is to use a technique called cross-compilation. The used framework provides an abstraction of the platform-specific features that can be accessed through an API by some mainstream programming language, such as Java or Ruby, that the developer uses for building the mobile application. The code is then translated by the cross-compiler provided by the tool into a native application that can be deployed and executed on the device. The code is often divided into business logic and user interface. In some cases it is easier to only cross-compile the business logic and implement the UI separately for each platform. Unlike hybrid applications that display a web page inside the app, this method has the look and feel of an actual native application, and access to all the same features. However, cross-compilers can be difficult to create and need to be kept consistent with the wide variety of mobile platforms and operating system versions available. It can be a challenge to find a cross-compiler tool that is kept up-to-date and provides access to all the latest features of the platforms, and they are often limited to only the most popular platforms. [26]

Unlike native applications that run directly on the device, interpreted applications use a virtual machine (VM) to abstract the target platform's features from the application's running code. This method installs a separate VM on the device, which not only provides the API but also the runtime environment in which the application will run on. The VM acts as a layer between the application and the operating system, and all the instructions pass through it. This tends to cause the application to run slower, but the benefit is that the VM is much easier to maintain and extend when new features are added to the operating system. The VM can be included in the application package, or it is downloaded and installed as a separate package.[26]

The next chapter will take a closer look at the individual cross-platform tools.

Chapter 4

Cross-Platform Development Tools

Developing mobile applications for several platforms with the native development tools typically means that the development cycle must be repeated for each individual platform. Each platform has its own family of devices, programming languages, APIs and distribution markets, and requires a specific set of skills to support it. As chapter 3 demonstrated, the mobile device user base is strongly fragmented between the different platforms, and choosing which platforms to support determines the amount of potential customers for the application.[19]

Different solutions to this problem have evolved over the years. Designing the application as a web application run in the native browser of the platform is a popular solution, but it suffers from limited access to the device hardware, and to some extent has a poorer performance and UI than a native application. Cross-platform tools seek to solve this problem by offering a way to package applications for different platforms from a common code base. The designing and development is done only once, producing a single code base. It can then be deployed to any supported platform by selecting target-specific customizations from the tool, and the application is distributed and installed on the device like a native application. Cross-platform tools generally rely on an abstraction layer between the shared application code and the target platform's native API, although the technical implementations and details vary between the different tools. Also, different tools require different skill sets and additional resources.[19]

Cross-platforms tools can be loosely divided into three groups depending on the level of functionality they offer: libraries, frameworks and platforms:[26]

- Libraries are small standalone toolkits that offer the developer functionality in a specific subject, such as 3D graphics or GUI controls. They are generally used in combination with other libraries and tools to create the full mobile application.

- Frameworks contain multiple libraries, software components and architecture guidelines for building entire mobile applications from start to finish.
- Platforms consist of a set of frameworks, tools and services that allow the developer to build and distribute mobile applications. They offer functionalities to debug, configure and package the application, and usually come with their own IDE, along with documentation, support and automation tools.

Figure 4.1 illustrates the market shares of different cross-platform tools in 2013.[40] Although there are dozens of tools available, only a handful are in wide use and they should be the main focus in this thesis. Also, some of the popular tools such as Unity 3D and Marmalade are primarily designed for game development and not suited for industrial applications. Next section will take a closer look at a few tools which are likely to be suited for ABB's use. The chosen tools are PhoneGap, Titanium, Qt, Xamarin and RhoMobile.

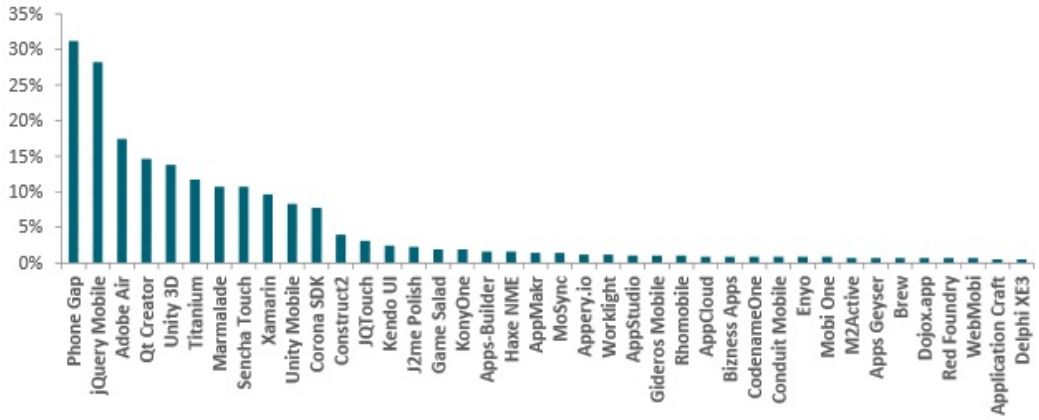


Figure 4.1: Usage distribution among cross-platform tool users in 2013, according to market analysis by Research2guidance [40]

4.1 PhoneGap

PhoneGap was originally created by Nitobi Software, which was acquired by Adobe Systems in 2011. It is an open source cross-platform mobile development framework under Apache License Version 2.0. PhoneGap can be used

for developing free, open-source or commercial applications. PhoneGap supports a wide range of mobile platforms, including Android, iOS, BlackBerry and Windows Phone.[34]

PhoneGap allows the developer to use web technologies, including HTML and CSS for the UI and JavaScript for the application logic, for developing hybrid applications that run in a WebView in a native application. As the UI is written using web technologies, the application has the look and feel of a web application unless extra measures are taken to mimic a native application's UI with platform-specific CSS files.[27] PhoneGap can also be used in combination with other development frameworks to improve the UI building. As PhoneGap itself lacks support for native UI components it is common to use tools such as JQuery Mobile or Sencha Touch to build the UI and use PhoneGap for the rest of the application and for wrapping the application in a native shell.[20]

Device's hardware functions are accessed through a JavaScript API provided by PhoneGap's engine. A table of the currently supported features for different operating systems is displayed in appendix A.5. PhoneGap can also be extended with plugins that grant access to additional features, such as Bluetooth[36]. As PhoneGap is an open source software, developers can also create their own plugins and features for it.[34]

PhoneGap does not come with a dedicated IDE so the developer can use any IDE for writing the code base. The code can then be ported to the target platform's own IDE for building, or built using the Cordova command-line interface. PhoneGap requires the native SDK of the target platform to build the final application. Adobe also offers the PhoneGap build service to build applications in the cloud. This allows the developer to build applications without installing the native SDK. [35]

4.2 Titanium

Titanium is an open-source application framework developed by Appcelerator and licensed under Apache Public License Version 2.0. It supports all the major mobile platforms, including Android, iOS, Windows Phone and BlackBerry.[7]

Titanium offers a different approach to cross-platform applications than most hybrid application tools. Instead of HTML and CSS, applications are written completely in JavaScript using the Titanium API. The code is packaged with Titanium's engine, which interprets the code at runtime and renders the UI. This allows the applications to have the look typical to native applications, as the UI is made up of native elements through the API.[27]

Appcelerator offers a free IDE for developing applications with Titanium, called Titanium Studio. The IDE works on all the major operating systems: Windows, Mac OS and Linux. Building the applications requires installing the native SDK on the system, so building iOS applications requires a Mac OS device.[7]

4.3 Qt

Qt is an application framework currently maintained by Qt Company, a subsidiary of Digia. It was originally released in 1995 by Trolltech, which was later acquired by Nokia in 2008. Nokia sold the commercial license of Qt to Digia in 2011 and Digia transferred Qt's copyrights to Qt Company in 2014. Qt is available as a commercial and open-source version. The commercial license includes additional functionalities, libraries and support, and full rights to modify the source code. The open-source version is available in different versions of the GNU Lesser General Public License(LGPL). Qt has traditionally been associated with desktop and embedded systems, but Qt 5.2 added support for creating Android, iOS and BlackBerry applications.[38]

Applications are written in standard C++ enriched with Qt's own macros. This makes porting existing desktop applications to mobile platforms using Qt especially easy, as desktop applications are often written in C++ and some of the business logic may be reused as-is. Several language bindings are also available for using other programming languages, including JavaScript, Python and QML, although they may lack some of the features.[38]

Qt's IDE is Qt Creator, which is included in every license option. It is available for all the major desktop operating systems and as usual, requires the native SDKs to be installed on the system as well.

4.4 Xamarin

Xamarin is a commercial application platform based on the Mono open source project, launched in 2001 by Ximian. Ximian was first acquired by Novell in 2003 and then by Attachmate in 2011. After the Attachmate acquisition, part of the original team that worked on Mono formed Xamarin, which would take over the Mono project. Xamarin currently supports iOS, Android and Windows Phone.[54]

Xamarin's main programming language is C# and with the release of Xamarin 3 also supports F#. Most of the code can be shared between the platforms, but the UI code is generally made separately for each platform

using the standard native functions and following the platform's UI conventions. The UI can either be done programmatically or by using Xamarin's graphical UI designer for the target platform. The latest version, Xamarin 3, also introduced the Xamarin.Forms that allows part of the UI design to be done platform-independently. This way the data on the screen is separated from the code that renders it, and only the renderers vary by platform.[10]

Xamarin offers developers its own standalone IDE Xamarin Studio. Alternatively, Xamarin can be integrated into Microsoft's Visual Studio. Building the applications requires the native SDKs.[54]

4.5 RhoMobile Suite

RhoMobile Suite is a commercial application platform currently developed by Motorola Solutions. It is based on the open-source Rhodes framework, originally released in 2009 by RhoMobile. Rhodes framework is available under the MIT license.[42] Commercial use requires the RhoMobile Suite, which includes an extended version of the Rhodes framework, called RhoElements. RhoMobile Suite has a subscription, which is available for free or as a paid enterprise version with additional features and support. RhoMobile supports iOS, Android, Windows Phone and desktop Windows.[41]

RhoMobile uses the Model View Controller (MVC) pattern for developing applications. It separates data definitions (models) from business logic (controllers) from interfaces (views) and provides the points that connect the three elements. The view elements are made using HTML, CSS and JavaScript, and the controllers use Ruby for backend support.[33]

RhoMobile Suite includes its own IDE called RhoStudio. RhoStudio is based on Eclipse and can use standard Eclipse plug-ins. RhoMobile can also be integrated into Visual Studio. Building applications requires the native SDK to be installed on the device, but the applications can also be built remotely using RhoHub.[45]

Chapter 5

Evaluation Criteria

This chapter establishes the criteria for comparing the different methods for developing mobile applications. The basic criteria are collected from other works in literature of comparing mobile development methods.[27] I then modify and expand the list by examining other aspects presented in literature and ABB resources and by having discussions with experts and developers. The criteria are divided into three categories: the criteria based on the end-user's experience, the development process and the business need.

5.1 End-user's Perspective

Here, the end-user is seen as the person who uses the final application. In the context of this thesis the end-user is typically ABB's own employee or an employee of the customer that has commissioned a product from ABB. From the end-user's perspective, applications that are installed on the device are expected to behave like native applications, regardless of their development method.[44] Therefore, how well the development method can match the native user experience and how much additional work is required to accomplish that goal are the main considerations. This is done from two angles, the UI look and feel and the application performance are examined separately. In addition, this section considers how well the different mobile platforms can be supported with the development method.

The criteria derived from the end-user's perspective are presented in table 5.1.

UI consistency with the target platform The end-user generally expects an application to behave a certain way on each platform. This includes aspects such as the style of the UI components and the location and behavior of buttons. From the application owner's side the application should be uniform between the platforms so that it is recognizable regardless of platform, and users can easily switch devices and still have the application behave the same way. This criterion examines how well the application matches the native look and feel and how much work is required for doing so.
Interaction and responsiveness Represents the overall speed and performance of the final applications. This includes for instance how quickly the application responds to inputs, moves between views and how well it can utilize the touch-based interaction model of mobile devices. Some cross-platform interfaces cause latency, because the commands must cross an additional abstraction layer. Also, some tools cannot utilize multi-touch functions without additional tools or libraries.
Supported platforms This criterion examines the range of platforms that are available for the method and how feasible it is to support multiple platforms with the method.
Energy consumption High memory inefficiency and mobile data usage increase the energy consumption of the application. While the application's actual context and implementation have the biggest impact on its energy consumption, the type of the application also affects it.

Table 5.1: Criteria based on the application's end-user's perspective

Shared code Reusing the same code for multiple platforms significantly reduces development time and makes targeting other platforms easier. This criterion examines how much of the code base can be common between the different platforms.
Access to native functionality Cross-platform development tools allow developers to use custom APIs for accessing the device's native functionality platform-independently. Therefore, the array of available native functionalities depends on the tool and how quickly it is updated when new features for the platform are released.
Competence availability This criteria examines the availability of the skill-set required for using the tool. Choosing a tool that uses uncommon programming languages or requires extensive framework-specific knowledge limits the number of available qualified developers. Training developers adds additional costs and slows down the development process.
Ease of design The design and prototyping process differs between the different methods. Quick prototyping improves the development process and allows the developers to easily share ideas and plans for the application.

Table 5.2: Criteria based on the developer's perspective

5.2 Development Perspective

This category examines how the choice of method affects the development speed and availability of proficient developers. Typically, the development process needs to be done separately for each target platform when developing native applications, and supporting an additional platform requires repeating the work. Using a cross-platform solution, some of the development effort can be shared between the platforms and only a small part of the application needs to be programmed separately for different platforms. The most common part that needs to be done separately for each platform is the UI design. Also, the programming languages and additional knowledge the tools require are examined, as training the developers for using a specific tool extends the development time. These criteria are listed in table 5.2.

5.3 Business Perspective

The development tool's effect on business is explored in the criteria presented in table 5.3. This includes aspects such as the financial implications of using

Cost This criterion examines the total cost of developing the application for multiple platforms using the method. This includes also subscriptions and additional software needed.
Long-term feasibility A mobile application in industrial use is likely to have much longer life-cycle than mobile applications have on average. Developing the application with a specific tool means that it needs to be supported and updated using the same tool, as porting it to some other tool is generally not possible or requires a heavy amount of manual work. This criterions examines the long-term feasibility of the tool based on how actively it is updated, how well it supports newest version of mobile platforms, and whether it has an active community and commercials supporters.
Publishing and distribution This criterion evaluates how easy it is to distribute the application to end-users and how well the update process works.
Security The choice of application type affects the security options available for it, such as encrypted data storage, secure authentication and access to other applications. Industrial applications are likely to play a critical role in large-scale systems and have access to sensitive information so good security is imperative.

Table 5.3: Criteria based on the business perspective

the tool to develop an application for multiple platforms and the possibility to support the application over a period of time. The section will also consider how easy it is to get the application and its updates to the end-user. Finally, any security issues with the development method are taken into consideration.

Chapter 6

Evaluation of Mobile Development Tools

The three different types of mobile applications examined in this thesis are native applications developed with the platform's standard development tools, web applications developed with standard web development methods and hosted as web sites, and hybrid applications created with cross-platform tools. There are differences in development and performance of hybrid applications depending on the toolkit used, but they are minor compared to differences between native, web and hybrid. Therefore, a single cross-platform tool is selected as an example and evaluated as the hybrid method, and the results will largely apply for other tools as well. PhoneGap will serve as the evaluated tool, because it is the most popular and promising cross-platform tool, and the tool supported by ABB's mobile development guidelines.

Each method for developing mobile applications is evaluated against the criteria set in the previous chapter. A written evaluation gives an overview of how well the method fulfills the given criteria and a numeric value is used for comparing the methods. The evaluation values are presented on a scale of 1 (very poor) to 5 (very good). The evaluations are based on other evaluations presented in literature and interviews with mobile development experts. The evaluations for individual methods are displayed in tables 6.1 through 6.6. Finally, a summary of the evaluation results, with the sum and average calculated for each method is shown in figure 6.1.

The evaluation arrives in the average ratings 3,8 for native, 3,7 for web and 3,5 for hybrid. Surprisingly, the values for the different methods are very close to each other. Native application has slightly higher rating than the other two, but only by a minor margin. Each method has its pros and cons and they suit different kinds of applications. Therefore, the target application needs to be taken into consideration in the selection process as

well. The application's requirements should be considered to select the best tool for each use case.

A second set of values is gathered in an online survey distributed to mobile developers, asking them to rate the methods by each criteria on a scale of 1(very poor) to 5(very good). The values from the survey answers are averaged and the results are displayed in figure 6.2.

The online survey produced similar results as the previous evaluation, with slight favoring of web application. The averages are 3,6 for native, 3,9 for web and 3,6 for hybrid. However, survey based evaluations tend to favor the middle values and do not highlight the differences between the methods. There are only a few cases of very high or very low values and different methods may get identical values for certain criteria, even though they are not equal in reality. For example every method had the average value 4 for ease of design, even though individual answers gave different values for the methods. The sample size was also too small to reach sufficient statistical power. Therefore using the survey results for choosing the best development tool is not suitable for the selection method used in this thesis. Determining the evaluations by a small group of experts was deemeed more suitable than averaging the results from a larger group of people. The initial evaluations will be used in the next step of the selection process, and the survey results show that the general trends of the initial evaluation are correct.

UI consistency with the target platform Applications developed with the platform's own development tools use the platform's native UI elements and can easily follow the conventions and guidelines of the platform.	5
Interaction and responsiveness Native applications offer the best performance and can utilize the full extent of the device's capabilities, such as multi-touch gestures and hardware buttons.	5
Supported platforms Every mobile platform has a method for developing native applications, so in theory every platform can be supported. However, the native methods only support their own respective platforms and the applications need to be developed separately for each platform. Therefore, native application approach does not support cross-platform development. Developing the application for an additional platform requires repeating most of the work.	1
Energy consumption Native applications require a lot of energy and a good network connection to be downloaded and installed. In actual use, the application's energy consumption depends on its features, but should stay low as long as the application has been developed properly.	5
Shared code The different platforms have their own programming languages and APIs and the code cannot generally be shared between them. Some common logic can be shared as modules by cross-compiling.	2
Access to native functionality Native applications have full access to the device's hardware and native features. They can access the platform's API directly and use the functions as soon as they are released.	5

Table 6.1: Evaluation of native application, first part.

Competence availability The programming languages used are fairly common, such as Java or C, but the languages vary by platform. This means that targeting multiple platforms with native applications requires a broad selection of skills. The developers also need platform-specific knowledge of the APIs and conventions. However, they are generally very well documented with numerous tutorials and examples.	4
Ease of design Designing a native application is fairly straightforward, as each platform has design guidelines that can be followed. Moving from the design to the actual running prototype of the application is quite slow for native applications. The UI needs to be constructed from scratch as the wireframes used in its design typically cannot be used in the actual application. Also, the application needs to be built and sent to a device or the developers have to rely on simulators.	4
Cost The native development tools are generally available for free, although some hardware restriction may apply such as needing a Mac to build iOS applications. Some platforms also require a developer membership for publishing applications to their app stores. For example, the Apple Developer Program has a subscription of 99$/year and registering as a Windows Developer costs 19$ for individuals or 99$ for a company. Also, as applications need to be developed separately, the development costs add up for each platform.	3
Long-term feasibility The current mobile platform market is quite stable, and the top platforms are likely to remain popular for several years. However, the smaller platforms can drop unexpectedly, which makes supporting them with native applications very risky. Also, new popular platforms may emerge very quickly, and the developers may need to act fast to support the new platforms.	4
Publishing and distribution Applications are distributed according to the platforms' conventions, usually through their respective app stores and for some platforms third-party sites. They must follow the platforms' policies and possibly go through an approval process depending on the platform.	3
Security Mobile platforms have powerful native functions for handling application security and native applications have full access to them. Applications generally run in a sandbox, and therefore cannot affect other applications. Local stored data can be encrypted.	5

Table 6.2: Evaluation of native application, second part.

UI consistency with the target platform Web applications have the look and feel of web sites, and look mostly the same regardless of platform. Imitating native UI requires using platform-specific CSS-files, causing additional work for each platform. However, as web application are accessed through the web browser instead of being installed on the device, the end-user does not expect them to behave like native applications.	3
Interaction and responsiveness Web applications' performance depends on the browser and internet connection speed. Mobile browsers are highly optimized, and while they cannot quite match the performance of native applications they are constantly getting closer. Web applications require a constant internet connection and disconnects can cause loss of data.	1
Supported platforms As mobile web application are web sites, they can be accessed with any web browser. Every mobile platform comes with its own native browser, and several more can be installed separately, such as Mozilla Firefox. Therefore, a single mobile web application works on every mobile platform.	5
Energy consumption Web application require constant network connection, which typically uses a lot of energy. However, unlike native and hybrid applications web applications do not have the same spike of initial energy consumption before they can be used.	2
Shared code The code can be shared completely between different platforms, unless platform-specific CSS-files are used. Browsers have minor differences, which may cause the application to look or behave differently on different browsers. This can be avoided with browser-specific code, but with the advancement of HTML5 and standardization of web development, it is rarely necessary.	5
Access to native functionality Web applications have very limited access to native functionality, and hardware features are typically inaccessible. The situation has improved with the introduction of HTML5, as many features that were previously not available were added, such a local cache for offline use and access to the device's camera and microphone. More features will be added in the future, but for now more advanced features, for example Bluetooth, are still not available.	1

Table 6.3: Evaluation of web application, first part.

Competence availability Developing web applications requires knowledge of HTML, CSS and JavaScript, and understanding of a mobile device's characteristics such as different screen sizes. Mobile web development is largely similar to conventional web development, so a web developer should be able to easily pick up mobile web development. Additional libraries have varying levels of documentation and support, so using them adds another learning curve.	5
Ease of design The application can be designed using wireframes that are written in HTML, and they can be used directly to build quick prototypes of the applications. Also web applications do not need to be built to be tested, making the entire development process faster.	5
Cost The cost of developing a web application depends on the used development environment and whether additional libraries are needed, but usually free ones are enough. Additionally, hosting the web site and any needed servers or databases add to the cost, but mobile applications are rarely stand-alone services so the infrastucture should be available anyway.	5
Long-term feasibility Web technologies are very well established and stable and are actively developed. Even if mobile platform market changes radically, web applications will likely work with the new platforms as well.	5
Publishing and distribution The application is published as a web site, and the user only needs the URL to access the application. Updating the application only requires the user to reload the page.	5
Security Web applications are typically more difficult to secure than native applications. The client side storage has weak encryption tools and applications are susceptible to the same web attack tactics as web sites in general are.	2

Table 6.4: Evaluation of web application, second part.

UI consistency with the target platform PhoneGap uses web technologies for rendering the UI of the application, causing it to have a web application look, which looks out of place in an installed application. Platform-specific CSS can be used to imitate the target platform's native look, but it requires extra work and still can rarely match the look perfectly.	2
Interaction and responsiveness PhoneGap applications' performance is almost comparable to native applications in light business applications with only minor latency, but applications that use heavy graphics can struggle. The WebView that PhoneGap uses is not as optimized as the mobile browser.	4
Supported platforms PhoneGap currently supports 10 different platforms, including all the major mobile platforms.	4
Energy consumption Like native applications, hybrid applications need to be downloaded and installed on the device. In use, hybrid applications are usually more energy heavy than native applications due to their extra abstraction layer and libraries.	4
Shared code The entire code base can be shared between the different platforms if the applications use the same UI. If the UI should mimic the native UI, the CSS files need to be made separately for each platform. Building with the native SDKs requires creating separate projects in each IDE and configuring the settings for the platform, but the main code can then simply be added to the project.	4
Access to native functionality PhoneGap's standard API grants access to the most common native functionalities and more can be added using plugins. As PhoneGap is open-source, anyone can develop new plugins if a functionality is missing or the existing API does not fulfill their requirements.	4

Table 6.5: Evaluation of hybrid application developed with PhoneGap, first part.

Competence availability Applications are developed with common web technologies: HTML, CSS and JavaScript. PhoneGap's APIs for accessing native functions are well documented and easy to use, so a web developer will easily be able to develop PhoneGap applications. Different functions are contained in their own plugins and only the ones that are needed for the application are imported into the project. Ready-made plugins exist for most of the common functionalities, but more specialized or optimized features may require developing a new plugin, which will also require knowledge of native technologies.	3
Ease of design As PhoneGap uses web technologies, some of the code used in building wireframes in HTML can be reused for the actual application, while making changes to fit the code into PhoneGap's API. Like native applications, hybrid applications need to be built and sent to a device to test. Hybrid applications have no clear guidelines to follow, and designing a suitable layout for multiple platforms is challenging.	3
Cost PhoneGap is open-source and allows creating commercial software for free, so the cost is the same for a single platform as for native applications. Supporting more than one platforms is cheaper as most of the development work can be shared between the platforms. Adobe also offers PhoneGap Build service for building the application in the cloud, avoiding the need for native SDKs and hardware.	4
Long-term feasibility PhoneGap is currently one of the most popular cross-platform development tools and has been stable for several years. It has an active developer community providing updates and new features, and its plugin-based architecture offers great flexibility. It also has the financial support of Adobe Systems. PhoneGap is likely to stay active for the foreseeable future, and in the worst case the application can be salvaged into a web application or another cross-platform tool that uses web technologies with a reasonable amount of work.	3
Publishing and distribution Applications developed with PhoneGap are distributed like native applications. PhoneGap applications have at times been rejected from Apple's App Store due to Apple's strict policy regarding applications that use primarily web technologies for not feeling native enough or being too slow. PhoneGap has released guidelines that should be followed when developing applications targeting Apple's App Store.	3
Security As the application is wrapped as a hybrid application and installed on the device, it has access to most of the same security features native applications have.	4

Table 6.6: Evaluation of hybrid application developed with PhoneGap, second part.

Criteria	Development tool		
	Native	Web	Hybrid (PhoneGap)
UI consistency	5	3	2
Interaction	5	1	4
Supported platforms	1	5	4
Energy consumption	5	2	4
Shared code	2	5	4
Native functionality	5	1	4
Competence availability	4	5	3
Ease of design	4	5	3
Cost	3	5	4
Long-term feasibility	4	5	3
Distribution	3	5	3
Security	5	2	4
total	46	44	42
average	**3,8**	**3,7**	**3,5**

Figure 6.1: Development method evaluation results based on literature and interviews with experts.

Criteria	Development tool		
	Native	Web	Hybrid (PhoneGap)
UI consistency	5	4	4
Interaction	5	3	4
Supported platforms	2	4	4
Energy consumption	4	3	3
Shared code	2	5	4
Native functionality	5	2	4
Competence availability	3	5	3
Ease of design	4	4	4
Cost	2	4	3
Long-term feasibility	3	4	3
Distribution	4	5	4
Security	4	4	3
total	43	47	43
average	**3,6**	**3,9**	**3,6**

Figure 6.2: Development method evaluation results from the online survey.

Chapter 7

The Tool Selection Matrix

The previous chapter showed that each of the mobile application development methods has its strengths and weaknesses and they suit different kinds of applications. The numeric evaluation gave very close average ratings for each of the methods, but the ratings for individual criteria show the differences between the methods. Therefore, the requirements of the application affect the choice as well, as different applications have different needs for the various features. For example, the need for the device's hardware functionality such as camera, storage or wireless communication depends completely on the intended use case of the application. For this reason, the selection process needs to take the target application into account. This is achieved by introducing weight multipliers for the individual criteria. This weight value depends on how important the criterion is for the target application, and is determined as 1 (not very important) to 3 (very important). Then, the matrix can include the weights as multipliers for the criteria and calculate a new, weighted average for each method. The results will determine which method is the best choice for that specific application. This extended matrix is named the tool selection matrix and is displayed in figure 7.4.

7.1 Target Applications

Three mobile applications in ABB are used as study cases for the selection process. Two of the applications already exist and will serve as examples for validating the selection matrix by comparing the method suggested by the matrix to the method that was actually used in developing the application. The third application is not yet in development and the selection matrix will be used to find the best development method for it. The features and the use purpose of the applications will determine the importance of the different

criteria for the application. The applications' actual names are not included in the thesis.

The first application will be known as application A. The application is used by both customers' and ABB's employees for managing and controlling drives. The application communicates with the drive via a two-way Bluetooth connection and collects data for diagnostics and fault analysis, and it can be used to set the drive's parameters and state. It also communicates with web resources and ABB databases. Figure 7.1 illustrates the communication methods of the application. The application is intended to be a complete package for managing drives with a large set of features and more are to be added in updates. The application was developed as a native application for Android and iOS. As the application requires Bluetooth support, a web application is not a possible option for it at this time. Therefore, the result for a web application is discarded for application A.

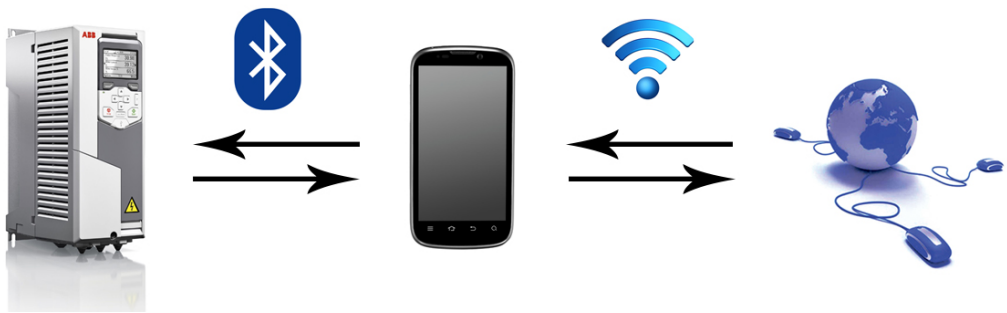


Figure 7.1: Application A's communication model.

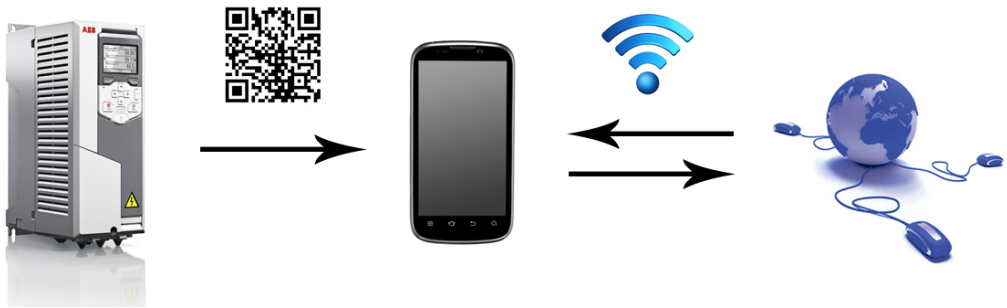


Figure 7.2: Application B's communication model.

The second application used for validating the matrix is application B. It is used by a customer to register their drive and get support. The user can

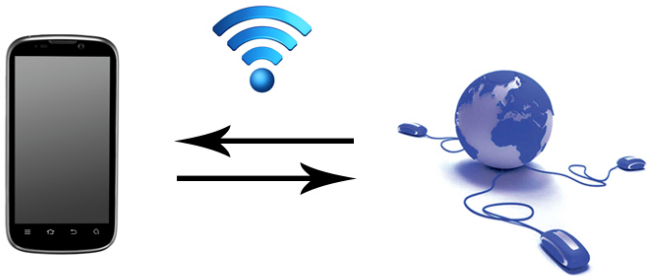


Figure 7.3: Application C's communication model.

scan the drive's serial number from a barcode or a QR code using the device's camera and then register the drive. The application also offers links to online resources, including manuals and customer support contact info, and can be used to receive maintenance recommendations for registered drives. It was developed as a hybrid application for Android and iOS using PhoneGap. Figure 7.2 shows the application's communication model.

A third application is also introduced, known as application C. It is an upcoming project in ABB and the result of the tool selection matrix will form a recommendation for a development method for it. The application is a mobile version of an existing web based service offered by ABB. It is used by a potential customer to find the best product setup for their need. The user selects the appropriate category of products and then fills in fields according to their use case. The application then calculates the products needed to match the use case and displays them in a list. The user can send the list to an ABB representative to receive an offer on it or save it to a file on their device for later use. The application communicates with ABB's web resources, as shown in figure 7.3.

The weight multipliers for these three applications and their reasoning are collected in tables 7.1 through 7.4. The weight values are gathered in interviews with the target applications' product owners. Each interviewee is asked how important each criterion is for the application on a scale of 1 (not very important) to 3 (very important) and why. The results are used as weight multipliers for the applications.

UI consistency with the target platform The application should follow the usual platform guidelines where possible, but the layout and UI features will be almost identical between the platforms.	2
Interaction and responsiveness Main focus for the application. Interaction needs to be smooth and have minimal latency.	3
Supported platforms The application needs to support at least the two major platforms, Android and iOS. The application will be used in an industrial environment by workers so they can be expected to have an Android, iOS or Windows Phone devices available, but they may have older versions of the devices. Also, the application should support both mobile phones and tablets.	2
Energy consumption Application will typically be used for only a short amount of time in one session, so energy consumption is not an issue.	1
Shared code As much of the codebase as possible should be shared between the platforms, and it should not be detrimental to the application quality.	3
Access to native functionality The chosen communication method with the drive requires access to Bluetooth functionality. In addition, camera and GPS should be available.	3
Competence availability The development is done by subcontractors so any additional training is to be avoided. The tool should already be familiar to the developers or require very little tool-specific knowledge.	2
Ease of design The application is developed with agile development methods and the tool should support quick prototyping and development. Also, the direction of the design may change during the development, so quick demonstrations are highly important.	3
Cost The development cost should be reasonable, while other criteria are fulfilled.	2
Long-term feasibility A drive is generally in use for 10 years or more, and the application should theoretically support the drive over its entire life-cycle. While this seems improbable for a mobile application, the tool should be chosen so that the application can be supported for as long as possible.	2
Publishing and distribution The application should be easily available to the customer. The application will have regular updates, and the possibility to distribute from a third-party site is considered a plus.	2
Security The application can be used to control the drive directly so security should be taken into account.	2

Table 7.1: Criteria weights for application A.

UI consistency with the target platform The application has a simple and clean UI. It should follow the platform-specific guidelines where possible, but its not the highest priority.	2
Interaction and responsiveness Good usability is one the main priorities for the application. Interaction should be smooth and have minimal latency.	3
Supported platforms The three main platforms, Android, iOS and Windows Phone should be supported.	2
Energy consumption Application will typically be used for only a short amount of time in one session, so energy consumption is not an issue.	1
Shared code Sharing some of the code is beneficial, but it is not the focus in choosing the development method.	1
Access to native functionality The application needs access to typical native functionalities, such as camera and GPS. Also the option to have access to Bluetooth in the future should be available.	3
Competence availability Proficient developers that already know how to use the development method need to be available.	1
Ease of design Design process is not a high priority.	1
Cost The development cost should be reasonable. However, there are enough resources available that it does not take priority over the other criteria.	2
Long-term feasibility The application is expected to stay active for several years and the development method should be able to support it over its entire life-time.	3
Publishing and distribution The application should be easily available to the customer, and will have some updates. The application uses external resources for storing large files such as manuals and user guides. They are stored on a web server and not included in the application package.	3
Security The user is required to create an account and supply personal information to register drives. The application needs to have authentication for the account and keep the information secure. It also has a connection to online databases.	3

Table 7.2: Criteria weights for application B.

UI consistency with the target platform The application is intended for personal use, so it should follow the platform conventions the user is familiar with. The application UI does not need to be identical between platforms.	2
Interaction and responsiveness All the heavy computing is done in the cloud instead of locally, so local performance does not need to be high. Filling the information should be smooth, but short loading time to get the end result is acceptable.	1
Supported platforms As the application is targeted toward potential customers looking to purchase drives and other products, it should be widely accessible. Not supporting the customer's device could cause a loss of customer. Therefore, as many platforms and versions should be supported as possible.	3
Energy consumption The energy consumption should be reasonable, but the application is not intended to be used continuously so it is not a high priority.	1
Shared code It is highly important that the different versions of the application targeted for different platforms have identical logic. The application is a sales tool and the information it offers needs to be up-to-date and consistent. Also, the application is targeted for multiple platforms and sharing the code between them will greatly reduce the development cost.	3
Access to native functionality The application does not need access to advanced native features. Possibility to access the user's contacts would be beneficial but not mandatory.	1
Competence availability Competent developers should be easily available when needed and the used technology well known. The application will not have constant development or a regular development team, instead a team is hired for a larger batch of updates at a time.	3
Ease of design The application should be stable, and only rarely receive updates. There needs to be a clear and easy way to design the application, as the developers may not be familiar with the application beforehand.	3
Cost The application is a sales support tool and is not critical for business. It is also an alternative version of an existing service, so the development cost needs to be low.	3

Table 7.3: Criteria weights for application C, first part.

Long-term feasibility The application is a stand-alone service and can be updated or replaced if needed.	1
Publishing and distribution The application needs to be easily available to end-users. The existing service only requires the user to visit a website and installing the mobile application on their device requires more effort. The user should also always have the latest version of the application, as an outdated one may have critical flaws. Different versions of the application need to receive the updates at the same time.	3
Security The data provided by the application is public information, so security is not an issue.	1

Table 7.4: Criteria weights for application C, second part.

7.2 The Selection Matrix

The previous chapter collected the evaluation results of the different methods into a matrix and calculated the sum and average of the results. Now, the weight multipliers of different target applications are added to the matrix, forming a tool selection matrix. Each criterion is multiplied with its respective weight, and sum of the multiplied criteria is divided by the sum of weight multipliers to get weighted averages. The final result shows how well the method fits for the target application. This process can then be repeated for each target application. The entire selection matrix with the added weight values and final results for the three example applications is displayed in figure 7.4.

The results of the two example applications: application A and B are examined first. Application A receives the ratings 3,8 for native, 3,6 for web and 3,5 for hybrid application from the matrix. The hard requirement for Bluetooth support means that a web application is not a feasible choice for application A and it is excluded from the results. In the current state of HTML5, a web application would not be able to provide the required native functionality. The ratings suggest that between a native and hybrid application, native is a slightly better choice. In this case, only a couple of platforms need to be supported so the main benefit of a cross-platform solution is diminished. Also, good user experience and interaction are the main focus points of the application, and a native application has a higher quality in these areas.

Criteria	Development tool			Weight values		
	Native	Web	Hybrid (PhoneGap)	Application A	Application B	Application C
UI consistency	5	3	2	2	2	2
Interaction	5	1	4	3	3	1
Supported platforms	1	5	4	2	2	3
Energy consumption	5	2	4	1	1	1
Shared code	2	5	4	3	1	3
Native functionality	5	1	4	3	3	1
Competence availability	4	5	3	2	1	3
Ease of design	4	5	3	3	1	3
Cost	3	5	4	2	2	3
Long-term feasibility	4	5	3	2	3	1
Distribution	3	5	3	2	3	3
Security	5	2	4	2	3	1
total	46	44	42			
average	**3,8**	**3,7**	**3,5**			
Weighted averages						
Application A	**3,8**	~~3,6~~	3,5			
Application B	**4,0**	3,4	3,5			
Application C	3,4	**4,3**	3,4			

Figure 7.4: The tool selection matrix

The benefit of using a cross-platform tool for developing the application A would be somewhat minor in terms of development time and cost saving, and it would risk compromising the quality of the final application. This reasoning supports the tool selection matrix's result, and application A should be developed as a native application separately for each target platform.

For application B the matrix produces the ratings 4,0 for native, 3,4 for web and 3,5 for hybrid. Native application has clearly the highest rating with a large margin. Once again, a web application would be a poor choice due to hardware functionality requirements. However, HTML5 has added some support for accessing the device's camera so it might be feasible to develop the application as a web application now, but the web application would still be an inferior choice to the other options. As the application is quite lightweight in terms of graphics and functionality, performance should not be a problem. The UI is simple enough that customizing it for different platforms would not require a lot of effort, and most of it can just be left the same for all platforms. The application was developed as a hybrid application with PhoneGap, and it would certainly benefit from hybrid application's cross-platform features such as shared code base and shorter development times. However, the matrix's results indicate that the application should have been developed as a native application instead. Like application A, application B's main focus is high quality user experience. Native application would have resulted in a higher quality final product, and the downsides like higher cost and longer development time were not given a high priority. The choice to use a hybrid application conflicts with the priorities set for the application. The implications of this result are further discussed in the next chapter.

The selection matrix has now found the best choice of development method for both of the example applications and they can be compared to the methods that were used in actual development. The result of the selection matrix matches the used development method for the first application, but differs for the second. The method suggested by the matrix would have been a better fit for application B than the one that was used based on the current priorities set for the application. These results verify that the method of setting application-specific weight multipliers for the different criteria and using the matrix to calculate the best development method is valid. The matrix can be used for any mobile application development project and the user only needs to think of the project's requirements and priorities and set the weight values accordingly, without needing knowledge of the individual development methods. The matrix then calculates the best fit for the project.

Having verified that the matrix works for existing applications, it can now be used to select the best method for an upcoming development project. The

third application already has its weight values in the matrix in figure 7.4 and the results can be read on the bottom row. The ratings are 3,4 for native, 4,3 for web which is the highest rating, and 3,4 for hybrid application for the application C. Clearly a native application would be a poor choice. The application needs to support as many platforms and versions as possible, to make sure a potential user's device is supported. Accomplishing this with native applications would simply not be feasible, the development cost would be too high and it would be near impossible to ensure that each version has identical and up-to-date logic. The application does not need advanced native functionality and does not prioritize user experience, so the main benefits of native application would be wasted. On the same note, there is no need to use a hybrid application, as its main advantage over web application is access to native functionality. A web application accomplishes everything that is required of the application. It can support a wide range of platforms with a single code base, with only minor modifications in UI code as per platform basis. It will also ensure that the user has the newest version of the application in use, and that the application has identical logic regardless of platform. It is also easily accessible for users and does not require installation. Based on these results application C should be developed as a web application.

Chapter 8

Discussion

As chapter 6 showed, each of the development methods has its strong and weak points, but on average they are almost equal. Therefore, they all exist for a reason. It would be very difficult to call one method the best one for every situation, so instead they need to be analyzed on a case by case basis. Adding multipliers to the criteria depending on the application being developed emphasizes the differences between the methods by prioritizing the features that are important for that specific use case.

The application type needs to be decided quite early in the design phase of the application and cannot easily be changed later. The designer might not have extensive knowledge of the different methods and how well they work for different situations, and choosing the optimal method is challenging. The main benefit of the selection matrix approach is that once the matrix has been established, using it for a new application allows the designer to focus on the needs of the application instead of the different development methods. They can set the appropriate weight values and the selection matrix will then match the best tool for the application with relatively little work. The weight values for a new application should be decided in the early design phase by the application owner. They should also be revised before starting the actual development, as the method most likely cannot be changed after that time.

The results for application A and C matched the initial expectations. Application A was developed as a native application, and the matrix recommended native application as well. Application C was planned as a web application and the matrix confirmed that this is the best choice. The result for application B was unexpected. The application was originally made as a hybrid application with PhoneGap, but the matrix suggests that a native application was a better choice. The choice to make a hybrid was made based on the lower cost and faster development time and the competence that was available at the time. Based on those priorities hybrid would indeed seem

the best choice. However, the priorities have shifted since the application was developed. The current view places priority on high quality user experience and usability. Things such as sharing a common codebase or available competence are not as important now. In retrospect, developing application B as a native application separately for each platform would have been a better option. The development method cannot be changed anymore at this point, but the result offers insight for future projects. It also demonstrates that the selection matrix does indeed find the best development method once the priorities have been identified.

The tool selection matrix will need to be updated regularly as the methods mature. Experiences with the online survey as discussed in chapter 6 showed that gathering data from a large group of people and averaging the results does not highlight the differences between the methods. Therefore, the suggested process is to update it once per year by a small team of experts. Each will review the evaluations and decide if any ratings should be changed. They can also consider if the criteria are still valid or if some should be removed or added. Then, they will discuss their findings in group and update the selection matrix accordingly.

Of course, the selection matrix has its shortcomings. The ratings will need to be updated regularly as the development methods evolve or the results will be outdated. Also, the list of criteria presented in this thesis has been created with industrial applications in mind. Using it for other kinds of applications, such as games, may require modifications. The rating approach does not process hard requirements well, such as needing a certain native feature. Therefore, it is necessary to manually prescreen the methods to identify impossible solutions. Then, the selection matrix would only be used for the remaining methods. This was done for application A, which needs Bluetooth support. An alternative solution could be to introduce a minimum threshold for certain criteria.

The selection matrix presented in this thesis only included one hybrid method. PhoneGap was seen as the most interesting and widely used hybrid development method and served as an example of hybrid methods in the selection matrix. Other hybrid development methods could be a subject for future study. They can be added to the selection matrix by following the process presented in this thesis and then the tool can be used for comparing the different hybrid methods as well.

The hybrid method's ratings on the matrix follow an interesting trend. It has the middle rating for most of the criteria, almost never being the best or the worst. This demonstrates how a hybrid application is the compromise solution. The cross-platform benefits come with a cost to performance and user experience. A hybrid application can reach very good quality with

enough effort and knowledge, but it is still very difficult to match a native application. For some applications the downsides are insignificant or there are too many platforms to support with native applications with a reasonable amount of effort, and in these cases a hybrid should be chosen for the reduced development costs and time. But if the resources are available to develop a native application for each target platform instead, most of the time it is the better solution. Perhaps when the hybrid methods mature, their downsides will diminish and they become more feasible alternatives for native applications.

Ultimately the context of the application plays the biggest role in choosing the best development method. Generally, a web application is the cheapest and fastest option, but it does not offer the same performance and functionality as a native application. For lightweight business applications that do not require advanced native functionalities a web application is a good choice. Using a hybrid application gives access to native features, but comes with its own downsides to look and feel of the application. It is a good alternative when native functionalities are required meaning a web application is not an option, and the application does not aim for the best user experience or is simple enough that there is no difference between native and hybrid. Also, if the application needs to support a high number of platforms, hybrid application becomes a better choice as native's application's cost climbs too high. The first two applications examined in this thesis needed to only support two or three platforms, so they would not have benefitted from hybrid application as much. Native applications tend to have the highest quality, offering a smooth user experience, familiar UI look and feel and the full range of the device's capabilities, but they need to be developed separately for each platform. The features of the application, available resources and the number of platforms that need to be supported determine if a native application is worth the extra effort.

Another consideration in the industrial setting is that the drives and other devices that the application needs to work with have much longer life-cycles than typical mobile applications. A drive can expected to be in use for 10 years or more, and during that time the entire mobile landscape is likely to change dramatically. New platforms will emerge and old ones may fall out of popularity, and the communication methods may change. The future trends should be kept in mind especially when designing the communication between a drive and a mobile device. Updating a mobile application or even replacing it is a relatively simple and cheap process, but updating the actual drive hardware is much more difficult.

8.1 Answering the Research Questions

The first chapter of this thesis set three research questions to aid in the study of mobile development methods:

- What are the key differences between the mobile application development methods?
- How do the required technical features limit the choice of mobile application development tool?
- Is a cross-platform solution a viable alternative to developing separate applications for each platform?

The key differences between the development methods are the aspects in the criteria list presented in chapter 5, and the choice between methods is made based on those criteria. Therefore, the key differences between the development methods are UI consistency with the target platform, interaction and responsiveness, supported platforms, energy consumption, shared code, access to native functionality, competence availability, ease of design, cost, long-term feasibility, publishing and distribution, and security.

For the second question the required technical features determine if the application is possible to develop as a web application and to a lesser extent a hybrid application. There are limitations on the availability of certain advanced device functionalities for web and hybrid applications. Needing a feature such as Bluetooth limits the choice to either a native or a hybrid application.

The answer to the third question is yes, a cross-platform solution can be viable option depending on the features and context of the application. The tool selection matrix presented in this thesis can determine if a given application should be developed separately for each target platform as a native application, or if a cross-platform solution would be suitable, either as a web or hybrid application. Examples of both cases are found in the study case applications. For application A and application B a native application is the best choice. Application C would benefit from a cross-platform solution and it was determined that it would be best to develop it as a single web application that can be used on any platform.

8.2 The Industrial Internet

Mobile applications working with industrial machines such as drives ties closely to the idea of Internet of Things, and its extension the Industrial Internet. Internet of Things is a vision of complex systems that combine hardware, sensors, data storage, microprocessors and software. At the core of this development is a "smart object" which is a combination of a physical component, such as a drive or a motor, and a computer that can process sensor data and support a wireless connection link to the internet. The object has a unique identifier, and it can be accessed and controlled from a distance. The ideal of Internet of Things is a large number of smart objects that can communicate with each other and other systems over the internet.[13]

Key component for the Internet of Things is the unique identifier associated with the object. In most cases this is accomplished with a small Radio Frequency Identification (RFID) tag. The reduction in terms of size, weight, energy consumption and cost of the radio component makes it possible to attach a RFID tag to virtually any object. This is combined with a sensor network that can constantly monitor the status of the objects, such as their location, temperature, movement and so on. This enables mapping the real world into the virtual world and monitoring it remotely.[11]

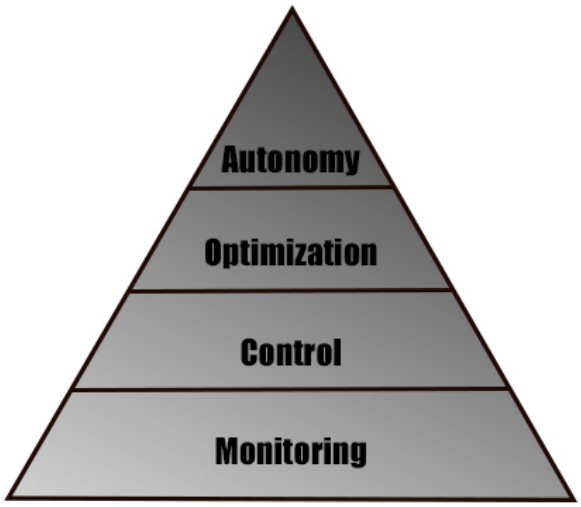


Figure 8.1: The capabilities of smart objects. Each layer builds upon the previous one.

The new capabilities offered by smart objects and the Internet of Things

can be grouped into four layers, each building on the previous one. The four levels of capabilities are monitoring, control, optimization and autonomy, illustrated in 8.1. The basic functionality of a smart object is the possibility to monitor the object's condition, operation and environment data through sensors. The gathered data can be used in a multitude of ways, such as reacting to changes in the object's environment or performance, or suggesting preventative maintenance. The second level allows remote control of the object, either through remote commands or automated algorithms. This enables a high level of customization for the object to best fit the use case. The third level uses the data gained from monitoring and the possibility to control the object remotely to optimize the performance of the object. The final layer is the dream of Internet of Things, where objects reach autonomy and carry out their tasks without the need for personnel. Examples of autonomy can be found even today in small scale, such as an automatic vacuum cleaner that can adapt to different floor layouts with the help of sensors and software.[37]

The applications examined in this thesis already demonstrate some of the capabilities listed above, although they still require a mobile device in conjunction with the drive to function. From the applications that were described in chapter 7, application B is used for monitoring the drive and application A offers a way to control the drive remotely. Most likely in the future more of the logic is included in the drive itself and an application is more of a way of tapping into that information.

The Internet of Things has a lot of potential, especially in the industrial setting. But it is still in very early stage, with only a few actual working implementations. Whether the Internet of Things turns out to be everything that has been promised remains to be seen.

8.3 The Future of Web Applications

Web application technologies have improved tremendously in the recent years. The fifth revision of the HTML standard, HTML5 was first introduced as a public draft in 2008 and officially released in October 2014.[51] It added support for several native functionalities that were not previously accessible for web applications and introduced plenty of other improvements to the technology.[49]

World Wide Web Consortium (W3C) is constantly working to further improve web technologies. A good example is their project Mobile Web Applications EU (MobiWebApp). The project's key goals are to increase the features available to Web applications to match native applications, develop

a mobile-friendly testing framework, offer training in developing web-based mobile applications and increase the awareness of the potential that web technologies can provide. The project has already given developers access to device's camera, address book and agenda through the browser using JavaScript, and made it possible to modify the user experience based on the battery level or capabilities of the network.[22]

In addition to those mentioned above, dozens of device APIs are currently under development by various groups in W3C. A full list of features under development by W3C can be found at www.w3.org/Mobile/mobile-web-app-state/. The work of Web Bluetooth Community Group is of particular interest for this thesis. The group is working on a Bluetooth API to allow websites to communicate with devices in a secure way. The work is still in a very early phase, the group was formed in July 2014 and the first demo published in December 2014, but it gives promise that in the future mobile web applications can utilize Bluetooth connection.[50] Other interesting upcoming features include for example continuous camera stream, encrypted storage, Near Field Communication (NFC) support and web notifications.

The future of web applications is certainly looking promising. New device APIs are being added, and current issues such as low security local storage are being improved. While it is difficult to say whether web applications can ever surpass native applications, they are making a good effort to compete with them. Much depends also on the changes in the mobile platform market. If a few platforms continue to dominate the market there is less need for cross-platform applications. But if the market becomes more volatile, platform-independent web applications and other cross-platform solutions will become more attractive.

Chapter 9

Conclusions

The aim of this thesis was to study different methods for developing mobile applications for multiple platforms and find the best development method for different use cases. Three main development methods were identified. Native applications are developed for a single platform at a time with the platform's standard development tools and conventions. Supporting multiple platforms with native applications requires the application to be developed separately for each target platform. Web applications are developed with web technologies and hosted as a web site. A web application only needs to be developed once and any mobile device can access them with their standard web browser via an URL. The third method utilizes web technologies but the resulting application is installed on the device like a native application. This is called a hybrid application. They are developed with cross-platform tools such as PhoneGap and the main part of application needs to be developed only once and can be deployed to any platform supported by the tool. Some platform-specific development is usually necessary, particularly for the user interface.

The development methods were evaluated and compared to each other. The evaluation was done based on a list of criteria, which included 12 items: UI consistency with the target platform, interaction and responsiveness, supported platforms, energy consumption, shared code, access to native functionality, competence availability, ease of design, cost, long-term feasibility, publishing and distribution, and security. Each method was rated by each criterion on a scale of 1(very poor) to 5(very good). The average ratings for the methods were 3,8 for native, 3,7 for web and 3,5 for hybrid. The ratings were too close to each other to declare any method universally the best one.

The evaluations were collected in a matrix, and the matrix was expanded into a tool selection matrix by adding application-specific weight values to the criteria. These weight values represent how important each criterion is for a

given application and range from 1(not very important) to 3(very important). The tool selection matrix calculates weighted averages of the ratings and the results represent how well a method fulfills a given application's requirements.

Three applications were introduced as study cases, known as application A, application B and application C. Of these, application A is an existing native application, application B is an existing hybrid application, developed with PhoneGap, and application C is a planned application with no set development method yet. Appropriate weight values were set for each study case application. The results suggested that for application A the best solution would be native application, for application B native application as well, and for application C web application.

The study showed that a cross-platform solution can be a viable option instead of developing the application separately for each target platform. A web application was the best solution for one of the applications studied, and in general would be a good choice for lightweight business applications that do not need advanced device functionalities. Hybrid applications seem inferior to native applications in their current state, but can be an acceptable alternative if the budget is tight or the application is intended for several platforms and needs device functionalities that a web application does not have access to.

The tool selection matrix can be developed further. Additional cross-platform tools can be added to the matrix by following the process outlined in this thesis. Also, the evaluations should be revised as the development methods advance.

Bibliography

[1] ABB. ABB-yhtymä. URL http://new.abb.com/fi/abb-lyhyesti/yhtyma.

[2] Open Handset Alliance. Overview, . URL http://www.openhandsetalliance.com/oha_overview.html.

[3] Open Handset Alliance. Android Overview, . URL http://www.openhandsetalliance.com/android_overview.html.

[4] Open Handset Alliance. Industry Leaders Announce Open Platform for Mobile Devices, 2007. URL http://www.openhandsetalliance.com/press_110507.html.

[5] Android. Android Developer. URL http://developer.android.com/index.html.

[6] AndroidDeveloper. Distribute. URL http://developer.android.com/distribute/index.html.

[7] Appcelerator. Titanium. URL http://www.appcelerator.com/titanium/.

[8] AppleDeveloper. iOS Dev Center, . URL https://developer.apple.com/devcenter/ios/index.action.

[9] AppleDeveloper. iOS Technology Overview, . URL https://developer.apple.com/library/prerelease/iOS/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html.

[10] ArsTechnica. Xamarin 3 review: Making cross-platform mobile development painless. URL http://arstechnica.com/information-technology/2014/05/xamarin-3-review-making-cross-platform-mobile-development-painless/.

[11] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. ISSN 13891286. doi: 10.1016/j.comnet.2010.05.010.

[12] Valerie Belton and Jacques Pictet. A Framework for Group Decision Using a MCDA Model: Sharing, Aggregating or Comparing Individual Information? *Journal of Decision Systems*, 6(3):283–303, 1997. ISSN 1246-0125. doi: 10.1080/12460125.1997.10511726.

[13] Mordechai Ben-Ari. *Real-Time Systems*. 2009. ISBN 9781848823136. doi: 10.1007/978-1-84882-314-3_21.

[14] BL Berg and H Lune. *Qualitative research methods for the social sciences*. 2004. ISBN 0205318479.

[15] Bimal K. Bose. *Power Electronics and Alternating Current Drives*. 1986. ISBN 978-0136868828.

[16] S Brahler. *Analysis of the android architecture*. PhD thesis, Karlsruher Institut für Technologie, 2010. URL `http://www.it.iitb.ac.in/frg/wiki/images/2/20/2010_braehler-stefan_android_architecture.pdf`.

[17] Margaret Butler. Android: Changing the Mobile Landscape. *IEEE Pervasive Computing*, 10(1):4–7, January 2011. ISSN 1536-1268. doi: 10.1109/MPRV.2011.1.

[18] A Charland and Brian Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):0–4, 2011. doi: 10.1145/1941487.

[19] Luis Corral, Andrea Janes, and Tadas Remencius. Potential Advantages and Disadvantages of Multiplatform Development Frameworks - A Vision on Mobile Environments. *Procedia Computer Science*, 10:1202–1207, January 2012. ISSN 18770509. doi: 10.1016/j.procs.2012.06.173.

[20] Isabelle Dalmasso, Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. Survey, comparison and evaluation of cross platform mobile application development tools. *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 323–328, 2013. doi: 10.1109/IWCMC.2013.6583580.

[21] Ben Elgin. Google Buys Android for Its Mobile Arsenal, 2005. URL `http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal`.

[22] Mc Forgue and D Hazaël-Massieux. Mobile web applications: bringing mobile apps and web together. In *WWW '12 Companion Proceedings of the 21st international conference companion on World Wide Web*, pages 255–257, 2012. ISBN 9781450312301. doi: 10.1145/2187980.2188022.

[23] Nisarg Gandhewar and Rahila Sheikh. Google Android: An emerging software platform for mobile devices. *International Journal on Computer Science and Engineering (IJCSE)*, (12):12–17, 2010. ISSN 0975-2297. URL `http://www.enggjournals.com/ijcse/doc/003-IJCSESP24.pdf`.

[24] Gartner. Press Release: Gartner Says Worldwide Tablet Sales Grew 68 Percent in 2013, With Android Capturing 62 Percent of the Market, . URL `http://www.gartner.com/newsroom/id/2674215`.

[25] Gartner. Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments Are On Pace to Grow 6.9 Percent in 2014, . URL `http://www.gartner.com/newsroom/id/2692318`.

[26] Gustavo Hartmann, G Stead, and A DeGani. Cross-platform mobile development. *Tribal, Lincoln House, The Paddocks, Tech. Rep*, (March): 1–18, 2011. URL `https://wss.apan.org/jko/mole/SharedDocuments/Cross-PlatformMobileDevelopment.pdf`.

[27] Henning Heitkötter, Sebastian Hanschke, and Tim A Majchrzak. Comparing Cross-Platform Development Approaches for Mobile Applications. In *Web Information Systems and Technologies*, pages 299–311. 2012. doi: 10.1007/978-3-642-36608-6_8.

[28] HTML5test. How well does your browser support HTML5? URL `https://html5test.com/results/mobile.html`.

[29] IDC. Smartphone OS Market Share, Q2 2014. URL `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`.

[30] Markku Laine, Denis Shestakov, Evgenia Litvinova, and Petri Vuorimaa. Toward Unified Web Application Development. *IT Professional*, 13(October):30–36, 2011. doi: 10.1109/MITP.2011.55.

[31] Reto Meier. *Professional Android 4 application development.* Joh Wiley & Sons, 2012. ISBN 978-1118102275.

[32] Julian Ohrt and Volker Turau. Cross-Platform Development Tools for Smartphone Applications. *Computer*, 45(9):72–79, 2012.

[33] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. Comparison of cross-platform mobile development tools. In *2012 16th International Conference on Intelligence in Next Generation Networks*, pages 179–186. Ieee, October 2012. ISBN 978-1-4673-1526-5. doi: 10.1109/ICIN.2012.6376023.

[34] PhoneGap. PhoneGap, . URL `http://phonegap.com/`.

[35] PhoneGap. PhoneGap Documentation, . URL `http://docs.phonegap.com/en/edge/index.html`.

[36] PhoneGap. PhoneGap Plugins, . URL `https://build.phonegap.com/plugins`.

[37] ME Porter and JE Heppelmann. How Smart, Connected Products Are Transforming Competition. *Harvard Business Review*, (November), 2014. URL `http://www.socialplm.info/FileLibrary/Topics/HarvardBusinessReview/HBR_How-Smart-Connected-Products-Are-Transforming-Competition.pdf`.

[38] Qt. Qt. URL `http://www.qt.io/`.

[39] Sara Radicati. Mobile Statistics Report , 2014-2018. Technical report, The Radicati Group, Inc., 2014. URL `http://www.radicati.com/wp/wp-content/uploads/2014/01/Mobile-Statistics-Report-2014-2018-Executive-Summary.pdf`.

[40] Research2guidance. Cross Platform Tool Benchmarking 2013 Hidden champions of the app economy. Technical Report October, 2013.

[41] RhoMobile. RhoMobile, . URL `http://rhomobile.com`.

[42] RhoMobile. Rhodes, . URL `https://github.com/rhomobile/rhodes`.

[43] Amit Kumar Saha. What is Android. *Linux for you*, (January):48–50, 2008. URL `http://tailieuandroid.googlecode.com/svn-history/r8/trunk/Andoid--tech.pdf`.

[44] Ahmed Seffah and Javahery Homa. *Multiple user interfaces: cross-platform applications and context-aware interfaces*. 2005. ISBN 978-0-470-85444-0. doi: 10.1002/0470091703.

[45] RhoMobile Suite. Documentation. URL `http://docs.rhomobile.com/en/5.0.0/home`.

[46] Taskumuro. Windows Phone 8. URL http://taskumuro.com/artikkelit/windows-phone-8.

[47] K. W. Tracy. Mobile Application Development Experiences on Apple's iOS and Android OS. *IEEE Potentials*, 31(4):30–34, July 2012. ISSN 0278-6648. doi: 10.1109/MPOT.2011.2182571.

[48] W3C. HTML5 Differences from HTML4, . URL http://www.w3.org/TR/html5-diff/.

[49] W3C. HTML5, . URL http://www.w3.org/TR/html5/.

[50] W3C. Web Bluetooth Community Group, . URL https://www.w3.org/community/web-bluetooth/.

[51] W3Schools. HTML5 Introduction. URL http://www.w3schools.com/html/html5_intro.asp.

[52] Anthony I. Wasserman. Software engineering issues for mobile application development. *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10*, 2010. doi: 10.1145/1882362.1882443.

[53] Windows. Windows Phone Dev Center. URL http://dev.windows.com/en-us.

[54] Xamarin. Xamarin. URL http://xamarin.com/.

Appendix A

First appendix

Mobile Application Development Survey

This survey compares different types of development methods for developing multi-platform mobile applications. The three methods included in the study are: developing the application separately for each target platform as a native application, developing the application as a cross-platform web application, and developing the application as a hybrid application using a cross-platform tool such as PhoneGap.

We ask that you rate each method by different criteria. Please consider how well the method performs on that criterion compared to the other methods. If you are not familiar with the method or otherwise have no opinion on certain questions you can skip them.

The survey has one page and should take no more than 10 minutes of your time.

The survey is a part of a Master's Thesis. Thank You for participating!

UI Consistency with the Target Platform

How well do you think this development method can match the UI style expectations of the end-user.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Figure A.1: Mobile development methods questionnaire, figure 1 of 4.

Interaction and Responsiveness

The overall speed and performance of the final applications. This also includes how quickly the application responds to inputs and how well it can utilize the touch-based interaction model of mobile devices

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Supported Platforms

Consider the number and importance of the platforms supported by the method.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Energy Consumption

The overall energy consumption of the application, including the downloading and installing of the application. Lower energy consumption is better.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Shared Code

How well the code can be reused between the different platforms.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Figure A.2: Mobile development methods questionnaire, figure 2 of 4.

Access to Native Functionality

The possibility and ease of accessing native functionality, such as camera, contacts or Bluetooth communication.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application					
Web Application					
Hybrid Application					

Competence Availability

Consider the skillset needed to develop the application for multiple platforms, and the availability of proficient developers.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application					
Web Application					
Hybrid Application					

Ease of Design

Rate the design process of developing an application with this method. Also consider how easy it is to make quick prototypes of the application.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application					
Web Application					
Hybrid Application					

Cost

The total cost of developing the application for multiple platforms using this method, including additional software and hardware needed. Lower cost is better.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application					
Web Application					
Hybrid Application					

Figure A.3: Mobile development methods questionnaire, figure 3 of 4.

Long-term Feasibility

How well the method works in the long-term. Is the method developed actively, how widely supported is it? How quickly can it react to changes in the mobile platform market distribution?

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Publishing and Distribution

How easy it is to distribute the application to end-users and how well the update process works.

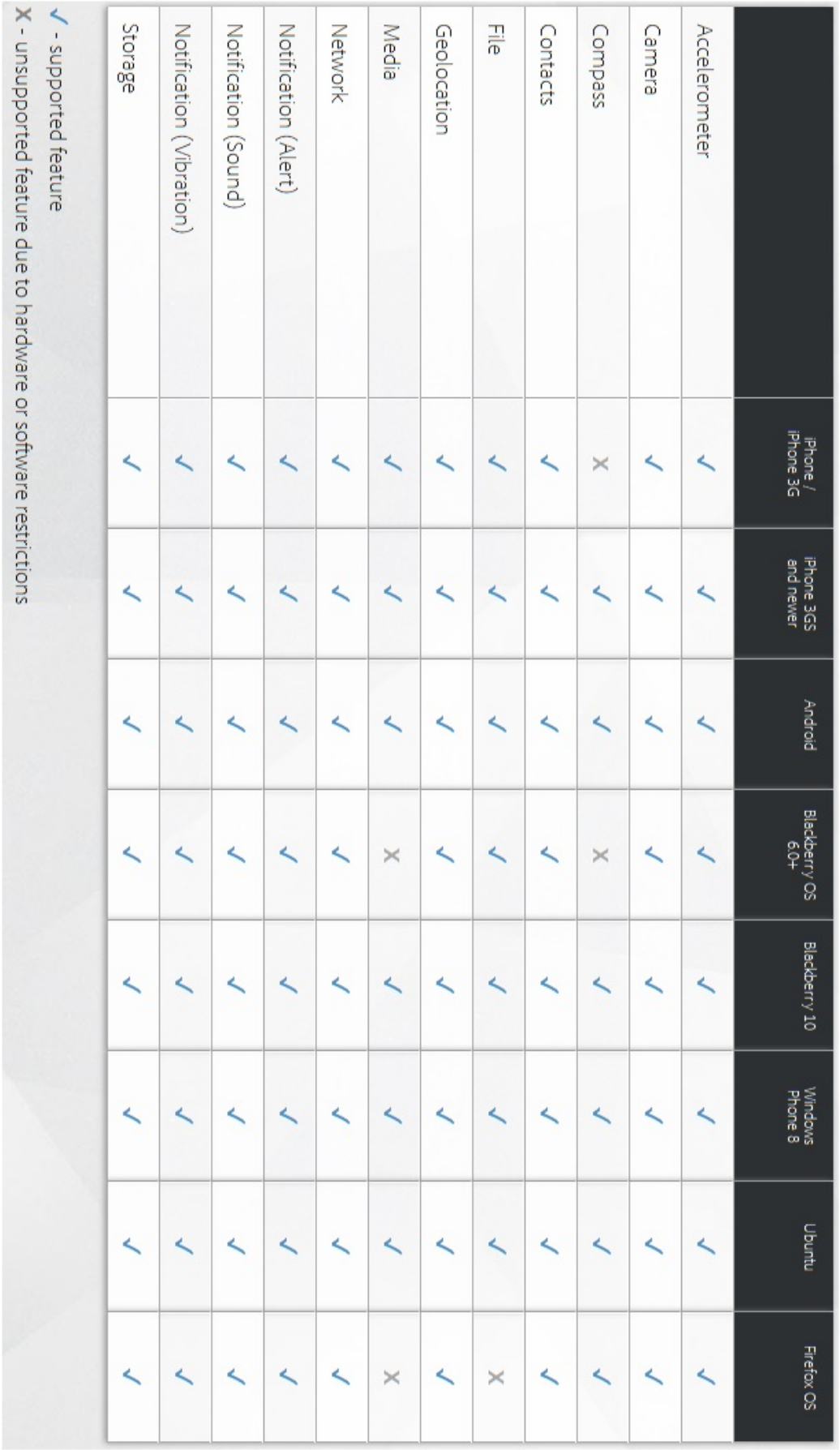
	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Security

Consider the security features available to the development method, and how vulnerable it is.

	1. Very Poor	2. Poor	3. Average	4. Good	5. Very Good
Native Application	○	○	○	○	○
Web Application	○	○	○	○	○
Hybrid Application	○	○	○	○	○

Figure A.4: Mobile development methods questionnaire, figure 4 of 4.



	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓
Compass	✗	✓	✓	✗	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓	✓	✗
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	✗	✓	✓	✓	✗
Network	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓

✓ - supported feature
✗ - unsupported feature due to hardware or software restrictions

Figure A.5: Features supported by PhoneGap.[34]