KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

A comparative study of cross-platform tools for mobile application development

Michiel Staessen

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Software engineering

Promotor:
prof. dr. ir. Erik Duval

Begeleider:
ir. Gonzalo Parra

Academiejaar 2012 – 2013

Preface

Michiel Staessen

Contents

Preface i

Abstract iv

1 Introduction 1

1.1 The mobile device landscape 1
Smartphones 1, *Tablets* 3

1.2 The problem of fragmentation 3

1.3 Cross-platform tools to the rescue 5

1.4 Goals 6

2 Literature study 7

2.1 Mobile application architectures 7
Native application 8, *Mobile web application* 8, *Hybrid application* 10, *Interpreted application* 10, *Cross-compiled application* 11

2.2 Multi-Criteria Decision Making (MCDM) 12
Arbitrary scoring models 13, *Analytic Hierarchy Process (AHP)* 14 , *Fuzzy MCDM* 17

2.3 Software evaluation methodology 18

3 Methodology 19

3.1 Define selection criteria 19

3.2 Identify potential candidates 20

3.3 List selected alternatives 20

3.4 Define evaluation criteria 22
Interviews with Capgemini employees 22 , *Literature* 25

3.5 Evaluate selected alternatives 27
Proof-of-Concept application 28 , *Evaluation methodology* 29

3.6 Select the most suitable alternative 30

4 About the evaluated tools . . . 31
4.1 Apache Cordova . . . 31
General information & history 31, *Supported platforms* 31, *Architecture* 32, *Related projects* 33, *Proof-of-concept application implementation* 34
4.2 Motorola Rhodes . . . 36
General information & history 36, *Supported platforms* 36, *Architecture* 36, *Related projects* 36, *Proof-of-concept application implementation* 38
5 Evaluation . . . 39
5.1 Weighting the evaluation criteria . . . 39
5.2 Evaluate the alternatives . . . 40
Platform support 42, *Toolset reuse* 42, *Code reuse* 44, *Access to hardware* 44, *Integration with platform-specific services* 46, *Native Look & Feel* 46, *UI element capabilities* 47, *Performance* 48, *Skill reuse* 49, *Tooling* 50, *Testing* 51
5.3 Costs versus benefits . . . 54
6 Conclusion . . . 56
6.1 Goals . . . 56
6.2 Reflection and future work . . . 57
A Requirements documentation of the proof-of-concept application . . . 60
A.1 Class Diagram . . . 60
A.2 Mock-ups . . . 60
Login screen 60, *Home screen* 61, *Wizard: Step 1 (Your Info)* 61, *Wizard: Step 2 (Overview)* 61, *Wizard: Step 2 (review expense from abroad)* 61, *Wizard: Step 2 (review domestic expense)* 61, *Wizard: Step 3 (add expense domestic expense)* 61, *Wizard: Step 3 (add expense expense from abroad)* 61, *Wizard: Step 4 (Sign & Send)* 61, *Wizard: Validation Dialog* 61, *Wizard: Validation element coloring* 61, *History* 61, *Smartphone version* 61
Bibliography . . . 62

Abstract

The `abstract` environment contains a more extensive overview of the work. But it should be limited to one page.

Chapter

1 Introduction

The mobile industry is without a doubt one of the most vibrant industries at the moment. It is characterized by rapid growth and intense competition which has led to fragmentation. This chapter first presents an overview of the evolution in the mobile device landscape, then explains the problem of fragmentation and how cross-platform tools (CPTs) can help solve this problem and finally, the goals of this thesis are defined.

1.1 The mobile device landscape

In this first section, the mobile device landscape and its evolution will be explored. It will introduce the concepts of smartphones, tablets and mobile operating systems and illustrate their popularity using worldwide sales numbers.

1.1.1 Smartphones

Mobile phones have been around for a while now but ever since the (nearly simultaneous) introduction of the iPhone 3G and the HTC Dream in 2008, smartphones are taking over from traditional cell phones or feature phones. A feature phone (sometimes also called a dumb phone) is a low-end mobile phone, providing only basic telephony and texting. Smartphones on the other hand are high-end devices that combine the functionality of mobile phones with the functionality of a portable computer. These devices have advanced computing power and often include multiple connectivity options (cellular, Wi-Fi, Bluetooth, NFC, etc.), sensors (GPS, compass, accelerometer, etc.), applications, etc. In the last five years, smartphone sales have grown tremendously. According to quarterly studies by Gartner[1] [1]–[16], smartphone sales have grown 544% since the second quarter of 2008 (see Figure 1.1). Smartphones are becoming ubiquitous and in some regions like the United States, smartphone penetration (this is the ratio between smartphones and all mobile phones) has already reached more than 50% [17].

Each of these smartphones comes with a mobile operating system which allows owners to run third-party software, typically called apps, on their device. These operating sys-

[1] Gartner is an American research and advisory firm, specialized in information technology, http://www.gartner.com.

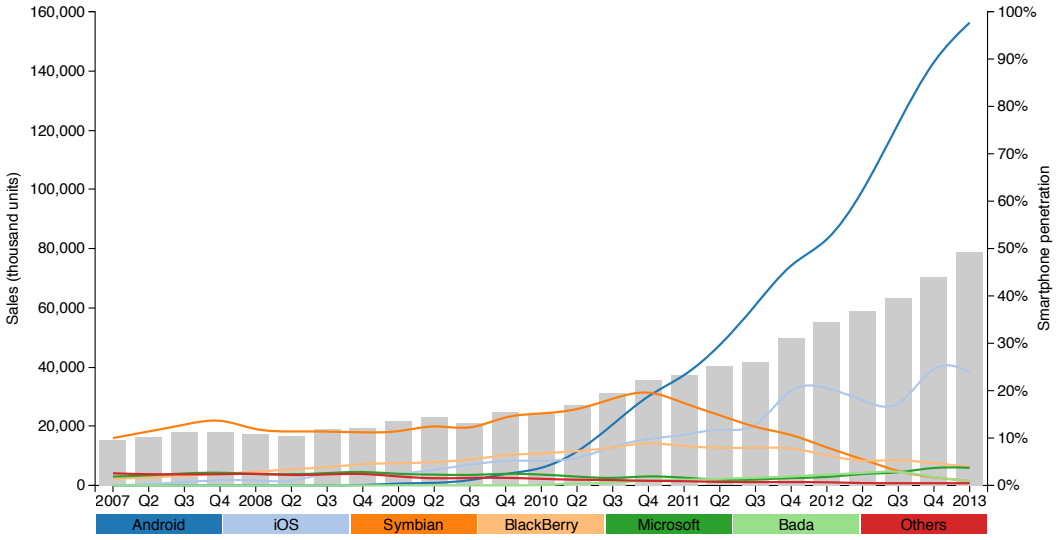

FIGURE 1.1: Evolution of worldwide smartphone sales by operating system (lines) and smartphone penetration (bars). Source: Gartner [1]–[16]

tems or platforms, are heavily subjected to network effects, i.e. their value is dependent on the number of people using it. This makes it hard for new platforms to gain traction which is visible in Figure 1.1 and Figure 1.2. Android and iOS are currently dominant and other platforms are either in decline (like Symbian and Blackberry, formerly RIM) or have a hard time getting traction (like Windows Phone, the successor of Windows Mobile, which already existed before Android and iOS).

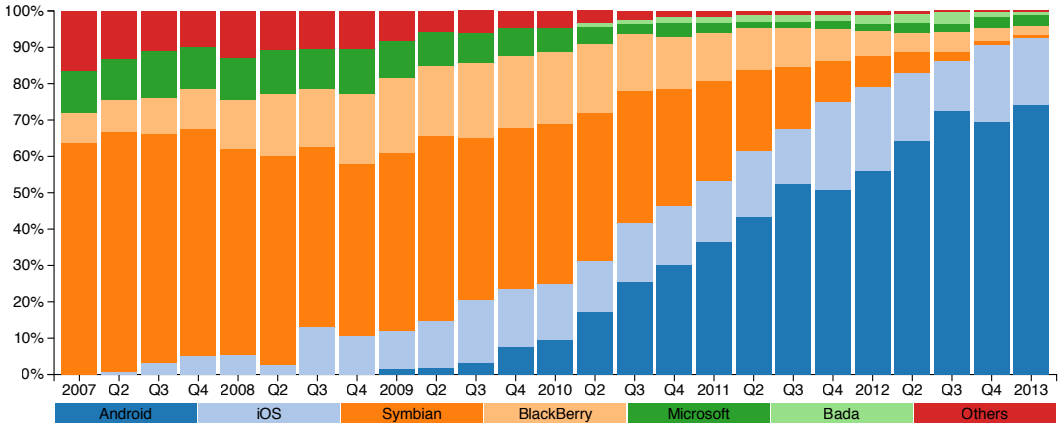

FIGURE 1.2: Evolution of worldwide smartphone market share by operating system. Source: Gartner [1]–[16]

However, there is no single major platform. In addition, the IDC[2] predicts that Windows Phone will gain a significant market share by 2016 and that 90% of the worldwide smartphone market will then be covered by Android, iOS and Windows Phone [18]. Hence, it is reasonable to assume that there will always be more than one major platform.

1.1.2 Tablets

The second type of mobile devices are tablet computers or tablets. In their current form, tablets are somewhat similar to smartphones but they have larger touchscreens (customarily starting at 7 inches in diagonal) and do not offer basic telephony. However, some of them do have a cellular radio that can be used for data transmission. Because of their hardware similarities, their software is also similar: the dominant smartphone operating systems are also used in tablets.

As with smartphones, tablets gained a lot of popularity since the launch of the iPad and Android tablets. According to other studies by both Gartner [19], [20] and the IDC [21], tablets will continue to gain popularity and sales will be mainly driven by iPads and Android tablets (see Figure 1.3). Even though these studies disagree on which operating system will be used in most devices, they both predict there will be three major platforms: iOS, Android and Windows.

TODO: Merge graphs into one image, this is UGLY!

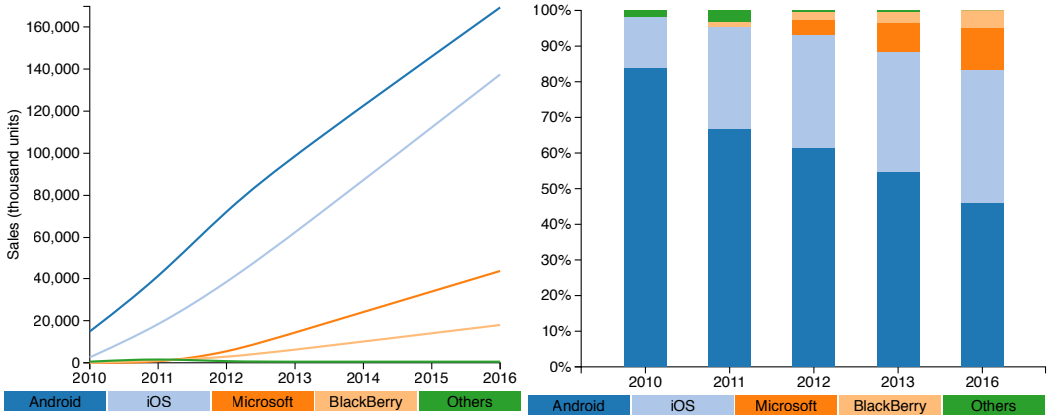

FIGURE 1.3: Prediction of worldwide tablet sales and market share. Source: Gartner [19], [20]

1.2 The problem of fragmentation

This section focusses on the problem of fragmentation. Fragmentation can be defined as the fact that all devices are different; there is no uniform device. This is called de-

[2] International Data Corporation is another American research, analysis and advisory firm, specializing in information technology, telecommunications and consumer technology, http://www.idc.com.

vice fragmentation. In fact, all mobile devices can be divided into multiple, overlapping categories like operating system or platform (called platform fragmentation), operating system version or runtime (called runtime fragmentation), screen size and screen resolution (called screen fragmentation) and many more. Hence, fragmentation is a multi-dimensional platform.

Fragmentation is generally beneficial for consumers, carriers and manufacturers. The more different devices there are, the more likely a consumer will find a device that fits his needs. For developers on the other hand, fragmentation is usually disadvantageous. It forces them to test their applications on multiple devices to guarantee the desired user experience. This is expensive and time-consuming.

The nature of a platform strongly influences the fragmentation issues within said platform. For instance, Apple can manage fragmentation issues pretty well because iOS is a closed platform and the only devices running iOS, called iDevices, are designed by Apple itself. In fact, these devices show many similarities. Android on the other hand is an open system. Android is being developed privately at Google, but the source code of every release is publicly available under the Apache 2.0 License, which means that everybody is allowed to customize it. The ability for manufacturers to alter the operating system has largely contributed to the success of Android. TODO: Update: On April 24 2013, the Google Play Store[3] listed 2827 officially supported device configurations, distributed across 59 manufacturers.

Mobile device manufacturers are eager to modify the Android operating system to differentiate their product from their competitors. As such, they create their own Android flavour, i.e. a distribution of Android with a custom user interface (like HTC Sense, Samsung TouchWiz, etc.), custom software, additional market places, etc. However, reapplying these modifications for every new release of Android is cumbersome and costly which is why device manufacturers do not often provide updates for their devices. This has led to the notorious Android runtime fragmentation, which is depicted in Figure 1.4. The adoption rate for new versions is low and is caused by the nature of Android. Developers have to support multiple versions, which is tedious.

Unlike Google, Apple does not publicize runtime statistics but developers [23] and online advertisers [24] can confirm fast adoption rates for iOS. On iOS, developers can make use of new functionality much faster. However, some devices that are still being used (like the original iPad and the iPod third generation) did not receive an update to iOS 6. Because of this, iOS developers now have to support the two latest versions of iOS.

Android and iOS are used in both smartphone and tablets. The number of different screen sizes and resolutions is low on iDevices. In contrast, there is an enormous number of screen sizes and resolutions on Android devices. This implies that applications also have to deal with various screen sizes and resolutions. On Android, this is solved by a flexible layout system. On iOS, this is solved by centering the user interface in the middle of the screen. For instance, applications that are not optimized for the 4-inch display

[3] Google Play is the official marketplace for Android applications.

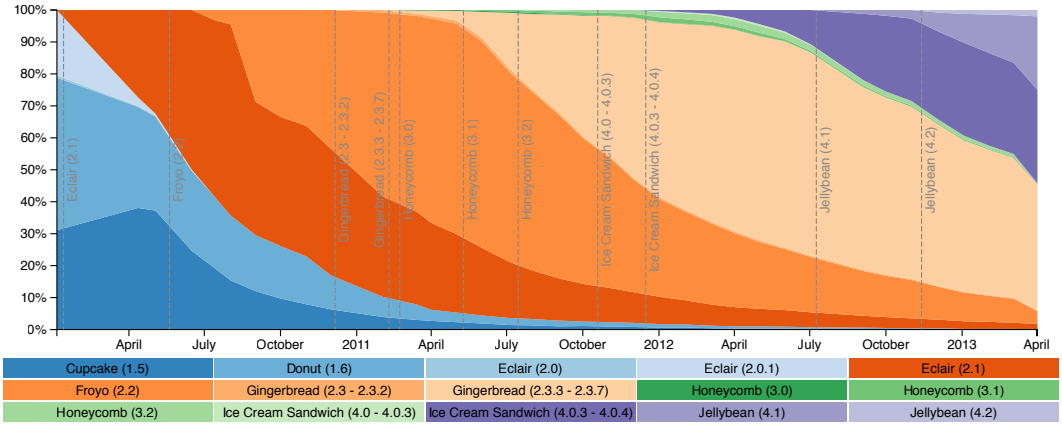

FIGURE 1.4: Historical Android runtime fragmentation. The data for this graph was aggregated from [22] using the Internet Archive (http://archive.org).

of the iPhone 5 (or iPod Touch, 5th generation) will have black bars on the top and the bottom. Retina displays do not really introduce a new screen resolution because every logical pixel is represented by four physical pixels and this conversion is mostly handled by the operating system.

There are still a lot more dimensions to the fragmentation problem like computing power, available sensors, available networking, etc. but these differences can be categorized under device fragmentation. In concluding, fragmentation among Android devices is rather high, fragmentation among iDevices is rather low.

1.3 Cross-platform tools to the rescue

In the current economy, information is a company's most valuable asset and the rate at which information exchange takes place increases every day. Mobile Internet-enabled devices are a valuable resource for this purpose and, as a consequence, many companies want mobile applications for their businesses. However, in an ever-changing and unpredictable industry like the mobile industry, it is unwise to target a single platform. This could eventually lead to vendor lock-in, i.e. all operations (or a significant part thereof) depend on equipment or software of a single vendor and switching vendors would require high costs. Companies will try to avoid these situations at all costs and they will do so by asking for cross-platform solutions.

These cross-platform solutions typically require that the software is implemented multiple times, one time for each platform. This is costly and time-consuming. Cross-Platform Tools (CPTs) can help to solve this problem because they allow to support multiple platforms from a single codebase. Hence, they lower entry barriers (access to new platforms) and exit barriers (lock-in) [25].

Cross-Platform Tools try to solve three major problems [25]:

1. **Fragmentation** Fragmentation issues as described above are a struggle for every developer. They are forced to test their applications on a large number of devices in order to be able to guarantee the desired user experience. A CPT can help to identify platform quirks and provide workarounds.

2. **Access to new platforms** Targeting a new platform is generally hard. Developers have to learn yet another SDK and/or programming language in order to deliver applications for this platform. A CPT can make abstraction of platform differences which allows developers to reuse their current skills. This drastically reduces the effort that is needed to target a new platform. Note that a new platform does not necessarily have to be a smartphone or tablet operating system, it could also be the operating system of a television set or a car console, etc.

3. **Development inefficiency** Maintaining codebases for multiple platforms is a difficult and costly task. When a new feature is introduces, it has to be applied to all codebases. With the use of a CPT, all code is contained within a single codebase and no time is lost while synchronizing features and other maintenance tasks across codebases. This reduces cost and increases productivity.

1.4 Goals

This last section defines the goals of this thesis. From the introduction, it is clear that there is a large demand for cross-platform solutions and that there are a lot of benefits associated with the use of cross-platform tools. Therefore, this thesis will try to identify a suitable cross-platform tool for mobile application development. This is a two-step process. First, a methodology will be defined for evaluating and selecting cross-platform tools. Second, this methodology will be used to evaluate a number of alternatives and to select the best-suited alternative.

TODO: maybe mention Cap?

Chapter

2 Literature study

This chapter presents an overview of literature that is related to this work. The first section will summarize the mobile application architectures that are currently used to create cross-platform solutions. In the second section, a generic software evaluation and selection methodology will be explored. In the third and last section, a number of techniques for multi-criteria decision making are studied, together with their strengths and weaknesses.

2.1 Mobile application architectures

Building cross-platform solutions starts at the architectural level. This section will discuss the architectures that are currently used with mobile applications or apps for short [26]. Mobile architectures comprise of three key elements:

- The *mobile device* is the piece of hardware that runs the actual application. This is the place where the architectures will differ the most.
- The *backend* is a service, provided by one or more servers that store all the application data. This could be a web service, ERP software like SAP, a CRM system like Salesforce, etc. or it could be a combination of such services.
- The *mobile hub* or *mobile orchestrator* is a server that acts as a mediator between the mobile device and the backend. It is put in place for multiple reasons: it exposes a uniform interface to the mobile device (and hides external API changes), prevents distributed denial-of-service (DDoS) attacks from the large number of mobile devices connecting to the backend, can cache information to reduce the load on the backend, can inject services like advertising into the application, etc.

The architecture of the native application is discussed first and is used as a reference architecture. For each of these architecture, a number of aspects will be highlighted: performance, look & feel, platform access, programming languages, development cost and distribution.

2.1.1 Native application

A native application is an application that is specifically designed to run on a particular platform. It is the default approach to develop applications for mobile devices and is therefore considered the reference architecture in this section. It is illustrated in Figure 2.1.

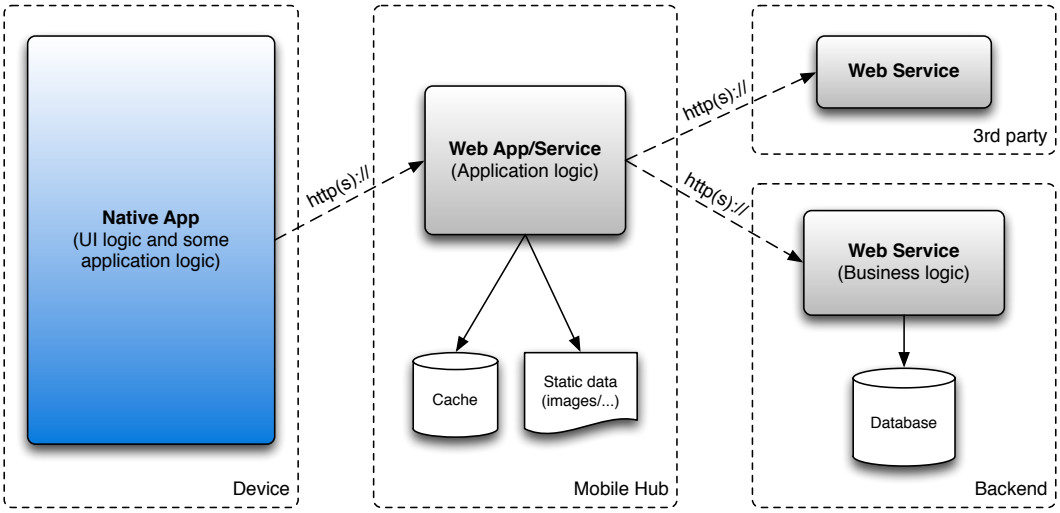

FIGURE 2.1: Architecture of a native application. Based on [26]

Native applications are developed with the supplied software development kit (SDK). That SDK uses a particular programming language that developers will have to learn together with the features of the SDK. In return they will get full access to the platform and its features. As a result, the best performance can be achieved with this kind of application.

The SDK also provides a large number of user interface elements that developers can use to create a rich user interface that is consistent with the look & feel of the platform. This is called native look & feel. Native apps are typically distributed through an online marketplace like the App Store for iOS applications or Google Play for Android applications.

The development cost of native applications is high because developers need to be familiar with multiple SDKs and the application has to be developed separately for every platform that has to be supported.

2.1.2 Mobile web application

Mobile web applications are websites that are optimized for mobile browsers. Since every platform comes with a browser, this is the easiest way to get an application running on all platforms. An overview of this architecture is depicted in Figure 2.2. Mobile web applications are built with HTML (content), JavaScript (behaviour) and CSS (styling, are

viewed in the browser and are distributed on the Internet with a URL but they cannot be installed on the device. A workarounds using Web Clips [27] is available on iOS and on Android, bookmarks can be placed on the home screen.

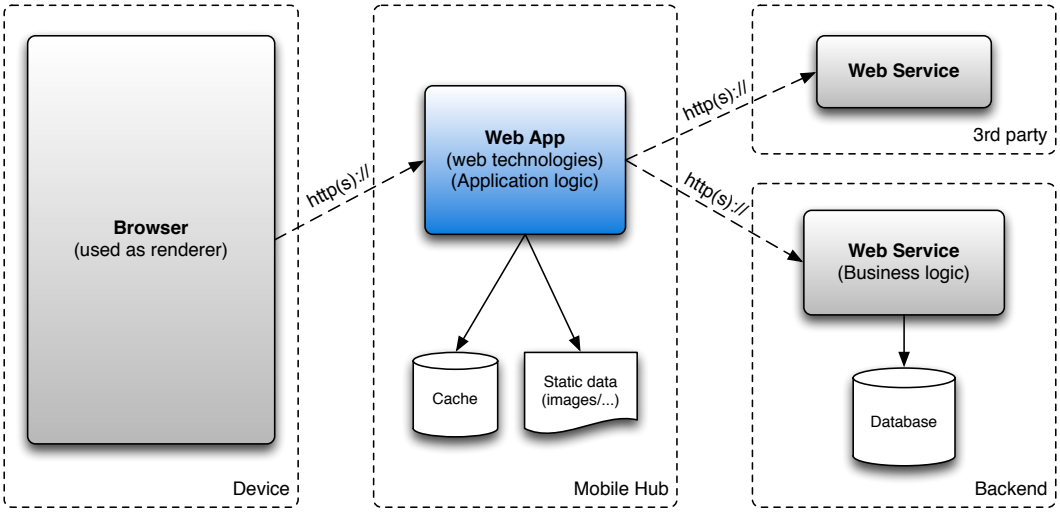

FIGURE 2.2: Architecture of a mobile web application. Based on [26]

Mobile web applications are not nearly as powerful as native applications because the web is currently not a first-class platform. Many mobile web browsers lack support for particular HTML5 features. These defects are listed on websites like "Can I use..."[1] and "Mobile HTML5"[2]. Developers should constantly check whether a certain feature is available. In some cases, the missing feature can be *polyfilled*. This term polyfill is inspired by the famous wall filler "Polyfilla" and is used for additional pieces of code that provide the missing HTML5 functionality. However, no polyfill can provide access to the underpinning hardware. Most mobile web applications also require an internet connection because the application is not cached in the browser.

Mobile web applications cannot make use of the native user interface elements. Because of this limitation, it is hard to provide a native look & feel. There are a number of project that try to mimic the native user interface elements with HTML templates and custom styles. However, the styling is quite complex and additional behaviour is added to animate these elements, which is disadvantageous for the performance. On the other hand, some people believe that the web is a platform of its own because it satisfies the "one size that fits all" philosophy. Therefore, it is entitled to its own look & feel [28].

The development cost of mobile web applications is low because it does not require additional skills (web development is often already in the skill portfolio) and the application has to be developed only once and can serve all platforms.

[1] http://caniuse.com
[2] http://mobilehtml5.org

2.1.3 Hybrid application

A Hybrid application is the combination of a native application and a mobile web application. The actual application is a mobile web application that is embedded in a native application and made visible through a WebView[3]. The native shell offers a JavaScript bridge to the web application, allowing the web application to access to the system. Because hybrid applications and wrapped in a native shell, they can be distributed through (official) marketplaces. The architecture is visualized in Figure 2.3.

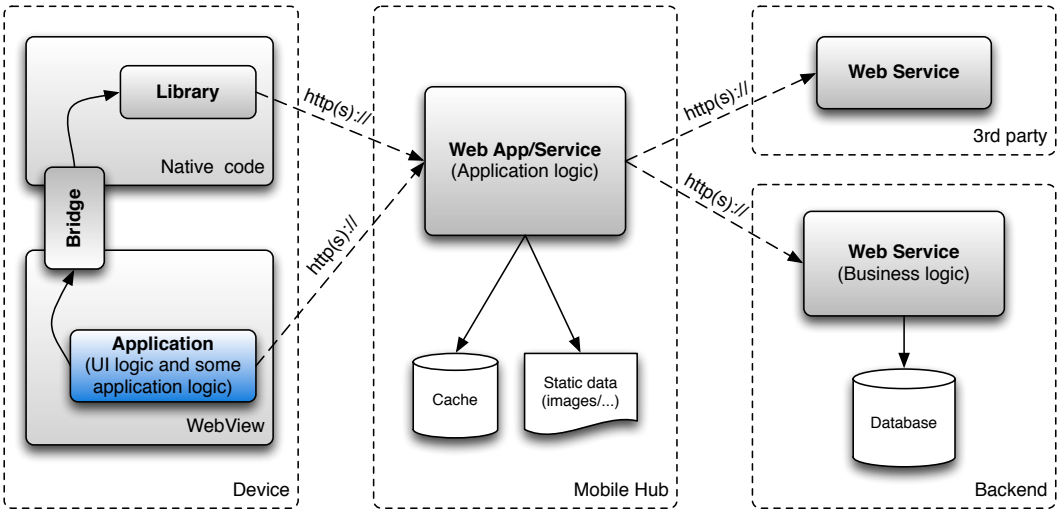

FIGURE 2.3: Architecture of a hybrid application. Based on [26]

Because of the nature of the application, the performance is be similar to web applications. Unlike mobile web applications, hybrid applications are more flexible because they allow better integration with the device through the JavaScript bridge. This way, missing features can be implemented natively and accessed through said bridge. Another consequence of its nature is that hybrid applications do not use the native user interface elements and cannot provide the native look & feel.

The development cost of hybrid applications is similar to that of mobile web applications because wrapping web applications does not require any knowledge of the native SDK's.

2.1.4 Interpreted application

Interpreted or runtime applications try to solve the platform differences by introducing an intermediary programming language. Instructions from that language are translated to native instructions at runtime. Interpreted applications are similar to Java applications on the desktop. From the outside, interpreted applications look just like native applications and can be distributed through online marketplaces. An interpreter or run-

[3] A WebView is a native user interface element that displays web content inside applications.

time is built right into the application, which increases the installation size of the application. This could be disadvantageous on low-end smartphones with limited storage capabilities. The architecture of an interpreted application is demonstrated in Figure 2.4.

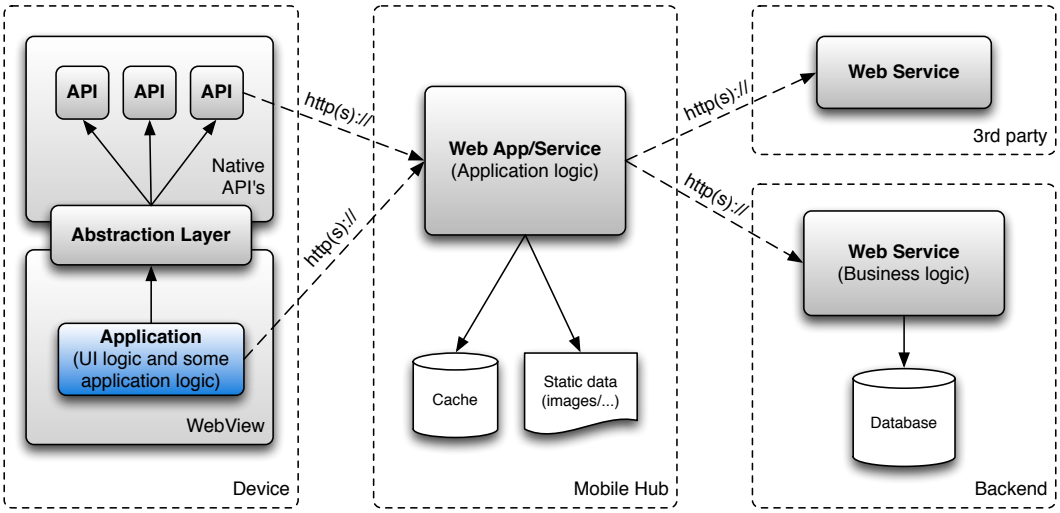

FIGURE 2.4: Architecture of an interpreted or runtime application. Based on [26]

The performance of an interpreted application strongly depends on the interpreter itself and the interpreted programming language. In general, the performance is better than the performance of web applications but it is not as good as the performance of native applications. Because of the nature of interpreted applications, the native user interface elements can be used to provide a native look & feel.

The development cost of interpreted applications is a higher than the cost of mobile web applications but lower than the cost of native applications. Developers need to be familiar with the SDK of the cross-platform tool but the application has to be developed only once and can be run on all supported platforms.

2.1.5 Cross-compiled application

In contrast to interpreted applications, instructions of cross-compiled are translated at compile time. The resulting application is not noticeably different from a native application because no runtime has to be included. This architecture is presented in Figure 2.5.

Because the application only exists of machine code, the performance of cross-compiled applications is nearly identical to the performance of native applications. Cross-compiled applications can make use of the native user interface elements and consequently provide the native look &feel. However, this sometimes requires custom bindings for each platform, increasing the amount of code that has to be written.

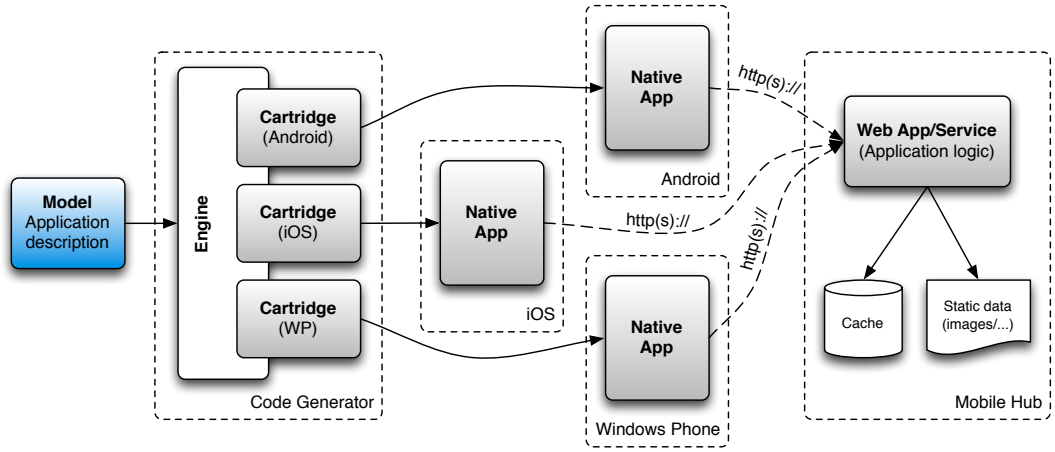

FIGURE 2.5: Architecture of a cross-compiled application. Based on [26]

The development costs of cross-compiled applications is similar to the costs of interpreted applications. Developers still need to be familiar with the SDK of the cross-platform tool but the application has to be developed only once and can be run on all supported platforms. Also, not all code can be reused in cross-compiled applications, which raises the development costs a little more.

Comparison of mobile architectures

To conclude this section, the discussed architectures are compared side by side. The results are shown in Table 2.1. Every architecture has both strengths and weaknesses. Therefore, the architecture must be chosen carefully, taking the client's wishes into account.

	Native	Web	Hybrid	Interpreted	Cross-compiled
Performance	high	low	low	average	high
Hardware access	✓	partial	✓	✓	✓
Look & Feel	native	non-native	non-native	native	native
Distribution	store	URL	store	store	store
Cost	high	low	low	moderate	moderate

TABLE 2.1: Summary of cross platform mobile application development strategies.

2.2 Multi-Criteria Decision Making (MCDM)

Finding the right software package is often a daunting task. In order to suit the end-user's needs, the software should meet a large number of – sometimes conflicting – requirements and will result in making important trade-offs. Because of the these characteris-

tics, software selection can be modeled as a Multiple-Criteria Decision Making (MCDM) problem [29], [30].

There are two categories of MCDM problems: Multiple-Attribute Decision Making (MADM) problems and Multi-Objective Decision Making (MODM) problems. The first category involves sorting and ranking of a limited number of available alternatives, based on a number of decision criteria. In the latter category, there are no alternatives specified beforehand and the number of alternatives is effectively infinite [31].

The software selection process belongs to the category of MADM problems. Their goal is to find the best alternative in a set of alternatives and at the same to time create a ranking of all these available alternatives TODO: .

There are a plethora of solution methods for the MADM problem. The following subsections will describe the most frequently used methods in literature, together with their advantages and disadvantages.

TODO: General remark for all methods: Separate costs from benefits, allowing to make a cost-benefit analysis afterwards.

2.2.1 Arbitrary scoring models

TODO: Question: Is this an appropriate name?

There are a number of arbitrary scoring models but they all have one thing in common: for each criterion, the values are translated to a numerical score. In some cases, this score can be derived from the value of the criterion itself (e.g. speed, age, cost, ...). In other cases, a mapping is provided (e.g. in order to obtain a score s, at least features f_1, f_2 and f_3 should be supported).

Different methods are available to select potential candidates using these scores [31]:

- When the "dominance" method is used, one alternative should clearly outperforms the other alternatives for at least one criterion.
- When the "maximin" method is used, the final score of each alternative is equal to the lowest score of all criteria for this alternative.
- When the "maximax" method is used, the final score of each alternative is equal to the highest score of all criteria for this alternative.
- When the "conjunctive" method is used, an alternative should exceed certain thresholds for *all* criteria.
- When the "disjunctive" method is used, an alternative should exceed certain thresholds for *at least one* criterion.

Additionally, the importance of the criteria can be accounted for by assigning weights. The final score can be calculated as the weighted sum, weighted product or weighted average.

The strength of these methods is that they are the most easy to use. However, scores and weights are assigned arbitrarily and might get tough when there are a lot of criteria. Also, not all criteria are suitable for conversion into a numerical scale [29].

2.2.2 Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) is "a multi-criteria decision making approach in which factors are arranged in a hierarchic structure" [32]. It is developed by Thomas L. Saaty and is based on pairwise comparisons of both criteria and alternatives.

The strengths of the AHP are (1) that it enables decision makers to structure a problem into a hierarchy, (2) that it provides a powerful tool for handling both quantitative and qualitative multi-criteria decision making problems and (3) that this system can deal with inconsistency [29].

The weaknesses of AHP are (1) that it is a time consuming method due to the large number of pair-wise comparisons and (2) that it is susceptible to rank-reversal, i.e. the ranking may change when new alternatives are added.

The analytic hierarchy process consists of three stages. In the first stage, the factors that are important for the decision are organized into "a hierarchic structure descending from an overall goal to criteria, subcriteria and alternatives in successive levels" [32]. Organizing the factors in such structure helps the decision maker in getting an overview of the potentially complex relationships and also helps to assess the relative importance of the issues in each level.

Structuring information in a tree is also backed by experiments of psychologist George Miller. He found that people can only deal with a few facts simultaneously; more precisely, seven plus or minus two **Miller:1956**

In the second stage, the analytic hierarchy process establishes a ranking of the alternatives. This ranking is based on (imprecise) judgements, resulting from pairwise comparison.

Consider n alternatives $A_1, A_2, \ldots, A_n$ and a property X that can be accurately measured for every alternative. The value of property X is given by w_i for alternative A_i. For instance, consider n stones with weights w_i, measured in grams.

The relative difference between all alternatives with respect to property X can now be

summarized in a matrix

$$A = \begin{bmatrix} w_1/w_1 & w_1/w_2 & \dots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \dots & w_2/w_n \\ \vdots & \vdots & & \vdots \\ w_n/w_1 & w_n/w_2 & \dots & w_n/w_n \end{bmatrix}. \tag{2.1}$$

Note that:

- A is reciprocal, i.e. $a_{ij} = 1/a_{ji}$
- elements on the diagonal are equal to one. This is easy to understand because there cannot be any difference between two identical objects.

If any row of A were to be multiplied by a vector containing all values $(w_1, w_2, \dots, w_n)^T$, the result would be a row of identical entries $(w_i, w_i, \dots, w_i)$.

In reality though, the values w_i are unknown and the ratios $a_{ij} = w_i/w_j$ cannot be calculated. However, an expert could estimate these ratios. The authors have developed a fundamental scale (see Table 2.2), which can be used to rate the difference between alternatives [32].

No human is perfect though and nor will be the estimates. The matrix $A' = \left[a'_{ij}\right]$ with estimated ratios will therefore be a small perturbation of A.

As a result, the aforementioned multiplication would not yield a row of identical values but instead it would yield a row of differing values, scattered around w_i (because of the errors in judgement). Consequently, it seems reasonable to represent w_i by the average of these values.

$$w_i = \frac{1}{n} \sum_{j=1}^{n} a'_{ij} w_j \quad i = 1, 2, \dots, n \tag{2.2}$$

This system of equations can be solved for $w = (w_1, w_2, \dots, w_n)^T$ if n is allowed to change. The problem now reads

$$w_i = \frac{1}{\lambda_{max}} \sum_{j=1}^{n} a'_{ij} w_j \quad i = 1, 2, \dots, n \tag{2.3}$$

which is equivalent to the eigenvalue problem $Aw = \lambda_{max} v$ and its solution is unique to within a multiplicative constant. Small perturbations of a_{ij} could generally lead to large perturbations of λ_{max} and w but it can be shown that this is not the case for reciprocal matrices **Saaty:1980**

The eigenvector w is normalized by dividing its entries by their sum to obtain the *priority vector*, containing appropriate weights or scores for all alternatives.

Intensity of importance on an absolute scale	Definition	Explanation
1	Equal importance	Two activities contribute equally to the objective
3	Weak importance of one over another	Experience and judgement slightly favour one activity over another
5	Essential or strong importance	Experience and judgement strongly favour one activity over another
7	Very strong or demonstrated importannce	An activity is favoured very strongly over another; its dominance is demonstrated in practice
9	Absolute importance	The evidence favouring one activity over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between adjacent scale values	When compromise is needed
Reciprocals of above nonzero	If activity i has one of the above nonzero numbers assigned with activity j, then j has the reciprocal value when compared with i. This is a reasonable assumption.	
Rationals	Ratios arising from the scale	If consistency were to be forced by obtaining n numerical values to span the matrix

TABLE 2.2: The fundamental scale for pairwise comparison[32]

Since this method is based on human judgement, which inherently contains errors, there needs to be a way to measure the amount of error in the result.

Reconsider A. This is a positive, reciprocal matrix and is also consistent, i.e. all entries satisfy the condition

$$a_{jk} = \frac{a_{ik}}{a_{ij}} = \frac{w_i}{w_k} \times \frac{w_j}{w_i} \quad i, j, k = 1, \ldots, n. \tag{2.4}$$

That is, all n alternatives form a chain.

The trace of a matrix, the sum of the elements on the main diagonal, is equal to the sum of its eigenvalues. Because all rows of A are linearly dependent, A has rank one. Consequently, all eigenvalues except one are zero. Hence, $tr(A) = n$, the order of A, is the largest or pricipal eigenvalue of A.

The amount of inconsistency, resulting from human judgement, can therefore be represented by $\lambda_{max} - n$, which measures the deviation of the judgements from the consistent approximation [32].

The authors define a *consistency index*, $C.I. = (\lambda_{max} - n)/(n-1)$. This consistency index is compared to the average consistency index of a large number of random reciprocal matrices of the same order, called the *random index (R.I.)*, listed in Table 2.3. When the *consistency ratio (C.R.)*, the ratio $C.I./R.I.$, is 10% or less, the estimate w is considered acceptable. Otherwise, new judgements should be collected in order to improve consistency.

n	1	2	3	4	5	6	7	8	9
RI	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45

TABLE 2.3: Random indices (R.I.) for reciprocal matrices of order 1 to 9

EXAMPLE TODO: Add example?

2.2.3 Fuzzy MCDM

TODO: Write section

Fuzzy logic was introduced in the mid '60s by Zadeh and was later applied to MCDM problems in the late '70s. Fuzzy logic allows to reason with imprecisely specified criteria like very high performance, low performance, etc. In classical approaches like those described above imprecision is dealt with first. When using the fuzzy approach, these imprecisions are only dealt with in the end, and only if necessary.

Fuzzy MCDM is based on three concepts **FuzzySetApproach**

1. The set of alternatives is a *fuzzy set*, i.e. the alternatives aren't precisely defined. The decision will thus be taken in a *fuzzy environment*.
2. The decision is based on an aggregation of fuzzy preferences among alternatives.
3. The decision is based on an evaluation of alternatives using a linguistic description.

More concrete, let $A = \{a_1, \ldots, a_n\}$ be a finite set of given alternatives and let there be m imprecisely defined criteria. Every alternative satisfies a criterion to a certain degree or does not satisfy it at all. In other words, for each criterion G_j, $j = 1, \ldots, m$ there is a set $G_j \subset A$ containing the alternatives satisfying the criterion and the degree in which an alternative a_i fulfills G_j is expressed by the membership degree $G_j(a_i)$.

In order to decide the MCDM problem, constraints must be defined for each criterion. These constraints are specified as fuzzy sets as well meaning that a constraint C_j is associated with a set $C_j \subset A$ containing the alternatives fulfilling C_j in a certain degree. Again, a membership degree $C_j(a_i)$ is given.

Consequently, there are two fuzzy sets for every $j, j = 1,\ldots,m$, namely G_j and C_j. The decision can now be obtained by composition and is given by the fuzzy set

$$D = (G_1 \alpha C_1)\beta \ldots \beta (G_m \alpha C_m) \tag{2.5}$$

where α and β are suitable fuzzy set operations. The alternative a_k with the highest membership degree $D(a_k)$ is the best alternative.

TODO: Add an example: For example, when buying a house these criteria could be *nice neighbourhood, great parking facilities, reasonable price*, etc.

2.3 Software evaluation methodology

Based on their literature review [29], the authors have proposed a generic, six-stage methodology for the selection of software packages [30].

1. **Define selection criteria.** In the first stage, the evaluator defines the essential requirements for the software. If a certain software package does not meet a selection criterion, it is not a considered to be a suitable candidate and it should not be considered for further evaluation.
2. **Identify potential candidates.** During the next step, the evaluator searches for potential candidates. This step will result in a list of potential candidates.
3. **List selected alternatives.** In this step, the evaluator will use the requirements from stagdate 1 to filter the list obtained from the previous stage.
4. **Define evaluation criteria.** In this stage, the evaluator has to define the evaluation criteria, arrange them in a hierarchy and define the value scales for each criterion.
5. **Evaluate selected alternatives.** During this phase, the evaluator will make a detailed comparison of the alternatives using the criteria obtained from the previous stage. A methodology like AHP can be used in this stage.
6. **Select the most suitable alternative.** In this final step, all alternatives are ranked using the comparison results from the previous stage. Now, the best alternative can be chosen from the list of candidates. In general, a number of software packages will be taken into consideration and a selection will be made after additional steps (like for instance a cost-benefit analysis and/or contract negotiations with the vendor).

In the original paper [29], the authors also suggest to include an additional evaluation stage after the selected packages has been implemented and integrated. During that stage, one should verify that the selected package does indeed meet the requirements.

Chapter

3 Methodology

This chapter describes the methodology used to compare and rank the studied cross-platform tools. The methodology follows the suggested 6-step approach presented in [30] (see chapter 2.3). The six steps are:

1. Define selection criteria
2. Identify potential candidates
3. List selected alternatives
4. Define evaluation criteria
5. Evaluate selected alternatives
6. Select the most suitable alternative

Every step will be further expanded in the following sections.

3.1 Define selection criteria

In this first step, the selection criteria for the tool are recorded. These requirements will be used later (in step 3) to filter a list of potential candidates.

As the research presented in this thesis is conducted on behalf of CapGemini, the selection criteria are determined by their consultants and were recorded during the kick-off meeting on October 15, 2012. In order to qualify as a viable cross-platform tools, it has to meet the following requirements:

- **It *must* produce "native" Android *and* iOS applications.** CapGemini focusses mainly on Android and iOS because their clients mainly focus on these platforms. Support for other platforms is desirable but not not a necessity.

- **It *must* be able to produce *both* tablet and smartphone applications, preferably from the same codebase.** Some clients want tablet applications, some clients want smartphone apps, some clients want both.

This list of essential requirements is extended with additional requirements. These are not essential as they can be circumvented in some way though they will generally result in higher productivity.

- **It *should* be usable to create enterprise applications with.** CapGemini specializes in the development of data-driven enterprise applications. Such applications usually contain a lot of forms and don't require high performance graphics (like for instance in 3D games). Even though it is possible to develop an enterprise application on top of a 3D engine, it will probably not result in good productivity.
- **It *should* have a certain degree of maturity** Ideally, the tool should be maintained for as long applications are created with it (and maintained).
- **It *should* have good support, provided by either the vendor or by the community.** In case of a problem, there should be a way to get support.

3.2 Identify potential candidates

In this stage, the evaluator tries to identify as much potential candidates as possible. These candidates do not necessarily have to meet the requirements from the previous stage as this is merely a discovery phase. The result of this stage will be a list of potential candidates.

Discovery of cross-platform tools has already been done extensively by VisionMobile. The latest cross-platform tools report [25] contains a list of 100 cross-platform tools they tracked as part of their research. Because additional internet searches did not reveal new tools, this list is used as output of this stage.

3.3 List selected alternatives

In this phase, the candidates obtained from the previous stage are filtered with the selection criteria from the first stage. The result of this stage is a list of alternatives worth investigating.

The list of tools from the previous stage contains a plethora of tools. Using the requirements from stage 1, a large number of tools can be left out:

- tools that do not produce native applications for Android and iOS;
- tools that do not produce tablet and smartphone applications;

- special-purpose tools that are not well suited for the intended use, e.g. specialized 3D engines;
- tools with an uncertain future, e.g. Flash-based systems or cutting-edge tools;
- tools that do not offer good support.

From 100 potential candidates, 7 tools listed in Table 3.1 were selected.

Name	Architecture	Type
Apache Cordova	Hybrid	Open Source
Appcelerator Titanium	Interpreted	Open Source
Motorola Rhodes	Interpreted	Open Source
Trigger.io	Hybrid	Commercial
MoSync	Cross-Compiled + Hybrid	Open Source
Kony	Hybrid	Commercial
Xamarin	Cross-compiled	Commercial

TABLE 3.1: The remaining tools after application of the selection criteria.

This thesis will compare two of these tools with each other and with the native development kits for both Android and iOS. The selected tools are Apache Cordova (formerly known as PhoneGap) and Motorola Rhodes.

Apache Cordova was chosen because of its popularity among developers and Motorola Rhodes was chosen because it focusses on enterprise applications.

Appcelerator Titanium and Xamarin were not selected because they are evaluated by another student with the same research topic.

Trigger.io was not selected because it was too similar to Apache Cordova at first sight: same architecture, same languages, ...

TODO: From what I read now, I should have selected MoSync, why didn't I? Don't really have a reason for it except maybe that I didn't research it quite well. Was also not prioritized by CapGemini but might have been my influence...

Kony — even though apparently using the the same architecture as Apache Cordova — targets enterprise applications, which made it a very good rival for Motorola Rhodes. However, there was no free trial available for Kony, which resulted in the selection of Motorola Rhodes.

Apache Cordova and Motorola Rhodes are discussed in more detail in chapter 4.

3.4 Define evaluation criteria

This is probably the most important of all stages. Setting up good grounds for comparison is paramount. The criteria are therefore carefully extracted from interviews with CapGemini employees and literature.

3.4.1 Interviews with Capgemini employees

A finished application is the result of a cooperation of different people with various roles. Every person experiences mobile application development in another way. In order to gain a better understanding of these experiences, three interviews with CapGemini employees were planned: one interview with a developer, one interview with a mobile architect and one interview with a sales representative.

The goal of these interviews is to reveal important criteria that should be considered during the evaluation of the cross-platform tools. The following sections present a summary of these interviews.

Developer

Developers are responsible for the development of the application. They can provide valuable insights into the typical technical issues that are encountered when developing a mobile application.

DEVELOPMENT ENVIRONMENT One of the relevant topics for application developers is the development environment since they will spend most of their time in it. Typically, the development environment is built on top of a Windows operating system and the Eclipse[1] IDE. This IDE is preferred because it is fast, efficient and highly customizable, which are desirable qualities for a good IDE. Also, most of the documentation is explicitly written for Eclipse.

DEVELOPMENT CYCLE Development happens locally and changes are deployed in an *unstable* environment. When features are ready to be tested, they are promoted and pushed to a *staging* environment, where they will be tested by for instance a number of end users. When the features are tested and ready to ship, a new production release is planned. In case mobile devices are involved, the application is tested on the device as much as possible.

The code is also checked overnight on a continuous integration server, which runs all unit tests together with a number of other useful tools like FindBugs[2]. Every morning, a detailed report of the state of the code is generated.

REQUIREMENTS FOR CROSS-PLATFORM TOOLS The requirements for cross-platform tools are that it (1) should work properly, (2) should not be too complex, (3) has at least decent performance (preferred over ease of use) and (4) that it is customizable graphically. Most clients have specific styling requirements.

[1] http://eclipse.org

[2] FindBugs uses static analysis to find common bugs, http://findbugs.sourceforge.net.

ATTITUDE TOWARDS NEW LANGUAGES AND TOOLS Learning new tools and languages is not considered problematic as long as it is worth investing in and/or requested by clients. Typically, one or more developers familiarize themselves with the tool/language and teach the other developers during a training session. Also, Capgemini has good partnerships to fall back on in case of a problem.

Mobile architect

The architect is responsible for the overall design of the application and makes the important decisions regarding used technology. Therefore, this person is best suited to describe the typical application structure and the trade-offs that have to be made constantly.

FOCAL POINTS IN MOBILE ARCHITECTURES Mobile architectures consist mainly of two major components: a front-end component and a back-end component. On the side of the front-end (the mobile application), the most important factor is the degree of integration with the device.

On the side of the back-end, the most important factors are the services the application needs to integrate with. An additional server, called the *mobile orchestrator* or *mobile hub*, will handle this integration and expose it in a single interface to the mobile application. This protects the back-end from the increased number of requests and allows for caching and other integrations like targeted advertisement campaigns.

REQUIREMENTS FOR CROSS-PLATFORM TOOLS There are a number of requirements for cross-platform tools:

- **Adoption** How many developers are using this tool? The number of users is mostly a good measure for the quality of the tool. A large user base is also beneficial when facing problems because it is more likely to get answers.
- **Future proof** Will the tool survive? Is there enough financial backing or community effort? It is not very wise to start a project that relies on a tool that has an unclear future.
- **Documentation** Good documentation is priceless as it can save numerous hours of digging through code.
- **Support** In case of a problem, is there an official channel to open a ticket? This could be either paid support or good community support. Either way, it should solve the problem. Delay due to unsolvable problems is unacceptable.
- **Customizability** Clients have specific requirements with regard to user interfaces and the tool should make this possible. Also, can the tool be expanded with plugins? A plugin architecture can contribute substantially to the customizability of the tool.

- **Supported devices** Which devices and runtimes does the tool support? In a B2B[3] context, devices are mostly controlled by an organization and the tool does not need to support many devices. In a B2C[4] context on the other hand, the organization cannot control the devices and consequently, the tool should support a lot more devices. Does the tool cover most or all of the used devices in this case?
- **Performance** The benefits of cross-platform development must justify the costs in terms of performance. If it took virtually no time to develop the application but it is slow, the application is useless.
- **Tools** Does this cross-platform tool allow to reuse the development tools that are already in use? Most people are more productive with the tools they are familiar with.

USABILITY Usability is a big concern when developing cross-platform. Users are accustomed to the native look & feel and wish to see new applications with similar styling. Even HTML5 applications are designed to mimic the native look & feel of the device. It is therefore hard to write applications with a single code base, even though this is desirable. Only when the application has a custom user interface that is in no way related to the native look & feel of the devices, the user interface can be reused across devices.

Sales representative

The sales representative records the client's requirements and sells the application. The salesman is therefore capable to describe the typical client requirements.

APPLICATION TYPES Roughly speaking, there are two types of applications: consumer applications and business critical applications. The first type of application often involves a lot of marketing and branding, which is not Capgemini's core business. For this kind of applications, businesses usually consult a marketing agency that makes this kind of (rather short-lived) mobile applications.

Capgemini focusses on the second type of application. These applications mostly handle data and do not rely on device specific hardware like for instance GPS or NFC[5]. However, clients are becoming aware of these device features and are starting to request this kind of device integration.

CUSTOMER REQUIREMENTS When clients come to Capgemini with a problem that they would like to solve with a mobile application, they typically do not have special technical requirements. These requirements are most often formulated in other terms. One example is the choice for native versus cross-platform. It could happen that the client has not yet decided on which devices to use or that the client does not want to create a lock-in situation by choosing for a single vendor. In that case, the application clearly has to work cross-platform. Clients could also wish to use NFC technology, which rules

[3] Business-to-business, or commerce between businesses
[4] Business-to-consumer, or commerce between a business and consumers
[5] Near Field Communication, a wireless technology based on RFID

out iOS as a platform. When there is no compelling reason to choose for either of both, Capgemini can help the client to make an appropriate decision. Performance is almost never a primary requirement. Perhaps this is implicitly assumed?

Summary of interviews

TODO: List of criteria resulting from interviews again?

3.4.2 Literature

The criteria extracted from the interviews are complemented with criteria from literature. As mentioned before in the literature study, only criteria that are used for evaluating benefits are selected here. By separating costs and benefits, a reliable cost-benefit analysis can be made in the end. The evaluation criteria from literature originate from reports by VisionMobile and Gartner.

VisionMobile[6] has conducted a worldwide survey about cross-platform tools. They received answers from over 2400 developers across 91 countries. The survey includes a question about the key reasons to select a particular tool and also includes a question about the key reasons to drop a cross-platform tool. The top 10 for both questions is summarized in Table 3.2 and Table 3.3.

Rank	Key reason to select tool	Share
1	It supports the platforms I am targeting	61%
2	Allows me to use my existing skills	43%
3	Low cost or free	40%
4	Rapid development process	33%
5	Easy learning curve	23%
6	Rich UI capabilities	19%
7	Access to device or hardware APIs	10%
8	High Performance / low runtime overhead	9%
9	Well suited for games development	8%
10	Good vendor support and services	8%

TABLE 3.2: Top 10 reasons to select a particular cross-platform tool. Respondents could enter their top 3 and only the response for the tools that are selected by 50+ developers are counted, which is 1512 responses [25].

It is clear from Table 3.2 and Table 3.3 that the key reasons for selecting a certain tool are more or less the same for dropping a tool. Common themes are (1) Platform support, (2) skill reuse, (3) development cost, (4) productivity or development time, (5) learning curve, (6) user interface capabilities, (7) device APIs and (8) performance. Non-common themes are (9) game development suitability, (10) vendor support, (11) tooling and testing and (12) concerns about the future of the tool.

[6] VisionMobile is an ecosystems analyst firm, http://www.visionmobile.com/.

Rank	Key reason to drop tool	Share
1	Low performance / high runtime overhead	29%
2	Does not support the platforms I am targeting	25%
3	Steep learning curve	24%
4	Restricted UI capabilities	22%
5	Does not let me use my existing development skills	22%
6	High costs	22%
7	Slow development process	21%
8	Challenges with debugging	18%
9	Access to device or hardware APIs	17%
10	Uncertainty of vendor future or platform roadmap	14%

TABLE 3.3: Top 10 reasons to drop a particular cross-platform tool. Respondents could enter their top 3 and only the responses for the tools that were dropped by 50+ developers are counted, which is 1109 responses [25].

From this list, (3), (4), (5) and (9) are omitted: (3) is clearly a cost, (4) is also a cost, albeit less clearly. However, developers need to be paid and consequently, development time is a cost. Capgemini does not develop games, which rules out (9). The interviews also revealed that typically somebody specializes in a certain tool or language and subsequently trains other developers and in case somebody gets stuck, there is almost always somebody to provide the necessary support. This should improve the learning curve and therefore eliminates (5) as an issue TODO: Is this valid argumentation? :/.

Gartner has also done similar research in 2011 [33]. The report of this research is not freely available but the criteria they identified are more or less clear from the table of contents. The criteria mostly overlap with the criteria from VisionMobile but also add (13) toolset reuse and (14) code reuse and organize them in three categories: portability, native experience and development experience.

Summary of literature

TODO: List the criteria from literature once more?

Summary of evaluation criteria

The identified criteria from the interviews and literature are combined into a single hierarchy of evaluation criteria. The criteria are organized in three categories, similar to the Gartner report.

TODO: I think I need to explain more why exactly these criteria are selected... After all, they are based on literature and interviews and I should point that out better...

- **Portability** describes the ability to reuse certain assets of a project when migrating to another platform.
  - **Toolset reuse** describes the reuse of development platforms like IDE's, operating systems, etc.
  - **Code reuse** expresses the amount of code that can be reused across platforms
  - **Platform support** describes the number of platforms that are supported by the cross-platform tool. For this criterion, only platforms other than Android and iOS are relevant as support for Android and iOS is a selection criterion.
- **Application Experience** describes the experience of the finished product: integration with the device, user interface and performance.
  - **Native Integration** describes the integration with native services.
    - **Access to hardware** describes the quality and extensiveness of device APIs.
    - **Platform-specific services** describes integration with platform-specific services like for instance Passbook[7].
  - **UI capabilities** describes the ability to create a rich user interface.
    - **Native Look & Feel** describes whether native user interface elements can be used.
    - **UI element capabilities** describes the capabilities of user interface elements.
  - **Performance** describes the overall performance of the application, like the snappiness of the user interface.
- **Productivity** is a measure for the ease of development.
  - **Skill reuse** describes the ability to reuse existing skills.
  - **Tooling** describes the quality of the development environment.
  - **Testing** describes the quality of software testing tools.

3.5 Evaluate selected alternatives

In this stage, the actual evaluation of the selected tools takes place. The evaluation is based on the AHP method, described in section 2.2.2. Therefore, all alternatives are compared in pairs with respect to the criteria recorded in the previous stage. In order to gain the experience that is needed to formulate accurate judgements, every tool is used to create a proof of concept application, or parts thereof.

The proof of concept application is described in section 3.5.1, the evaluation method is described in section 3.5.2. The actual evaluation of the tools is described in chapter 5.

[7] Passbook is an iOS application to gather electronic tickets in one place.

3.5.1 Proof-of-Concept application

The proof-of-concept application is a rather small but typical enterprise application. It is not typical in the sense that it "looks" like the average application. Instead, it is rather typical in the sense that is contains the most requested features. This helps to ensure that the selected tools are thoroughly tested with regard to these essential features. This section describes these features in detail. The requirements documentation is available in Appendix A.

Context & scenario

Employees of certain companies occasionally have to make costs for which they would like to be reimbursed. The process for this reimbursement typically involves keeping books, filling out forms and a lot of waiting while a superior deals with the request. Needless to say, there is a great potential for a mobile application here.

The application is designed to do just that. Employees can group a number of invoices into one request, provide evidence for the costs in the form of pictures, sign the document on a phone or tablet and send it to the backend. From there, the request is forwarded to a qualified person that will review the request and deal with it.

Typical functional elements

The proof-of-concept application includes a number of features that are typically required in any application.

- **UI elements** Most applications are useless without a user interface. This application incorporates a number of frequently used UI elements.
  - **Form elements** Virtually all input is captured with "forms". The form elements should support read-write and read-only mode. This application uses different types of form elements to represent different kinds of data.
    - **Text** For inputting arbitrary text.
    - **Number** For inputting numbers.
    - **Email** For inputting email addresses.
    - **Password** For inputting passwords, the content of the input field not shown as characters but as symbols.
    - **Drop-down** For selecting an item from a list.
    - **Radio button** Also for selecting an item from a list.
    - **Toggle switch** To toggle a state on a property, e.g. an on/off switch.
  - **Button** Used to trigger some action.
  - **Tab bar** Used to switch between contexts.
  - **Activity indicator** A spinning wheel that is displayed when the user is needs to wait for a while because the applications is handling some request.

- **UI modes** The application must support multiple screen modes.
  - **Tablet UI** The display mode used on tablets. The layout typically consists of a narrow column on the left and a wide column on the right (also known as the master–detail interface).
  - **Phone UI** The display mode used on smartphones. Nearly the same layout but shows less detail master and detail views are decoupled in separate views.
- **Serialization** In order to communicate with the backend, data must be serialized using various formats. This application makes use of XML, JSON and plaintext.
- **Input validation** To make sure that the data that is entered is consistent and valid, some validation needs to be done on the device (and on the server).
  - **Data type validation** If a field only accepts numbers, it should automatically make clear to the user that other input than numbers is invalid.
  - **Custom validation** There might be some restrictions on some fields that are dependent on other selections in a form. For this kind of consistency checks, custom validation is essential.
- **Sorting** Data-driven apps display a lot of data. It is easier for the end user if the data can be sorted in the right way.
- **Offline mode** The application should still work when the device is offline.
- **Local Storage** The application needs to be able to store some data that is either persistent or cached.
- **Session management** Business applications are typically bound to a user account. Hence, session management on the client side is also required.

Most of these commonly requested features can be built on top of other features but that requires extra work. A tool is more interesting if it includes this functionality out of the box. Part of the evaluation is to investigate the ability to use these functions.

3.5.2 Evaluation methodology

The Analytic Hierarchy Process is used to evaluate the alternatives. First, a weight is assigned to every criterion. This weight is the normalized principal right eigenvector[8] of the pairwise comparison matrix containing the criteria in a single category. The weight of each criterion is multiplied by the weight of its parent.

Second, a weight is assigned to every combination of alternatives and criteria. All alternatives are compared in pairs with respect to a single criterion and the results are

[8] In AHP, the length of a vector $\mathbf{v}$ is given by the sum of its entries, not by the Euclidian distance between $\mathbf{0}$ and $\mathbf{v}$.

combined in a pairwise comparison of which the principal right eigenvector can be calculated. Again, the weights are given by the normalized eigenvector.

Last, the total weight of every alternative is calculated to obtain a ranking of the alternatives. In case of n alternatives and m criteria, the total weight of an alternative A_j is calculated as

$$a_j = \sum_{i=0}^{m} c_i w_i \quad j = 1, \ldots, n \tag{3.1}$$

where c_i is the weight of the criterion C_i and w_i is the weight of alternative A_j with respect to criterion C_i. The alternative with the highest weight is "the winner". However, the winning alternative may have the most benefits but it could also carry the highest costs. To make sure that the costs are justified, an additional cost-benefit analysis is performed.

Because this is the most important part of this thesis and to long to fit in here, the evaluation is discussed in detail in chapter 5.

3.6 Select the most suitable alternative

The final step of the software selection process is the selection itself. The decision maker chooses the best-suited alternative based on the results from the evaluation. Since the goal of this thesis is to find the best alternative for cross-platform development, this step is discussed in the conclusion (see chapter 6).

Chapter

4 About the evaluated tools

This chapter provides elaborate descriptions of the evaluated tools and information about the proof-of-concept implementations.

4.1 Apache Cordova

4.1.1 General information & history

Apache Cordova[1] started out as a project called PhoneGap. It was developed by a couple of employees from a Canadian company called Nitobi and was unofficially released at the iPhoneDevCamp gathering in August 2008. In 2011, Adobe acquired said company such that the team could focus solely on the development of PhoneGap. At the same time, Nitobi donated the PhoneGap codebase to the Apache Software Foundation (ASF), where it became an incubating[2] project. To make sure that the intellectual property would not conflict with trademark ambiguity, the project's name was changed from PhoneGap to Apache Callback. Because the community was dissatisfied with the project's name, it was quickly changed to Apache Cordova. On October 2012, Apache Cordova graduated as a top level project within the Apache Software Foundation. This ensures that it will always remain free and open source under the Apache License 2.0.

Nowadays, PhoneGap is a distribution of Apache Cordova, delivered by Adobe. PhoneGap strategically fits in a collection of software like Adobe PhoneGap Build and Adobe Edge Inspect (see further). The development and maintenance of the cross-platform tool happens in the Apache Cordova project and is driven by an active community in which Adobe is the largest contributor.

4.1.2 Supported platforms

Apache Cordova supports a large number of platforms, both mobile and non-mobile.

[1] http://cordova.apache.org

[2] In order to become part of the ASF, every project is required to go through an incubating stage.

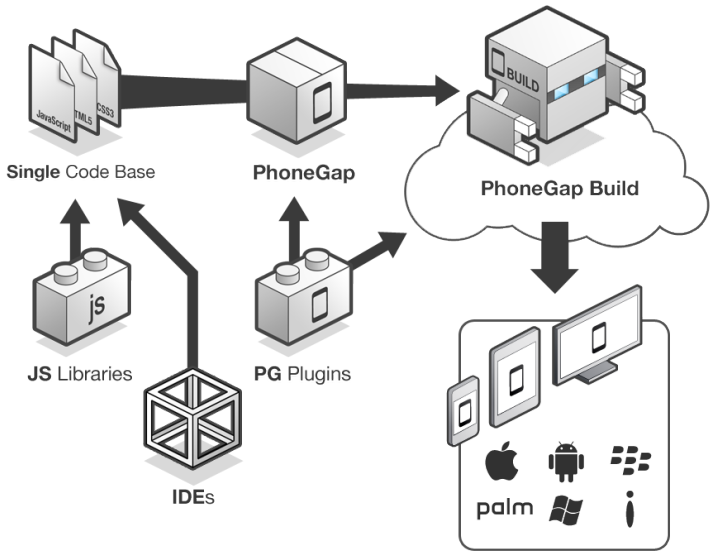

FIGURE 4.1: The architecture of Apache Cordova/Adobe PhoneGap applications.

- The current release supports Android (2.1 and up), iOS (4 and up), BlackBerry (5 and up), Windows Phone (7 and 8) and Windows (7 and 8).
- Future releases of Cordova are likely to support for Tizen, Qt, Firefox OS, Ubuntu Mobile.
- Support for Symbian, Bada and WebOS is decreasing as these platforms are considered to be dying.

4.1.3 Architecture

Apache Cordova is a cross-platform tool that produces hybrid applications, i.e. the actual application is a mobile web application. This web application is wrapped inside a native shell which provides access to the hardware through a JavaScript bridge. This architecture is visualized in Figure 4.1. Wrapping an application can be either done locally or through a building service in the cloud called Adobe PhoneGap Build.

One of Cordova's goals is to promote the web as a first-class development platform. However, many mobile browser implementations are still lacking decent HTML5 support. Cordova solves this problem by providing HTML5 polyfills for these browsers. A polyfill is a sort of plugins that addresses the lack of a specific feature in a browser. The API of the polyfill typically matches the official API such that applications do not notice that the feature is missing. If the browser does have support for a particular feature, the browsers implementation is used.

EXAMPLE The Cordova Storage API is a polyfill for the deprecated but often used Web SQL specification[3] and Web Storage specification[4].

Cordova has a pluggable architecture, i.e. within Cordova, every device API is implemented as a plugin. This allows to (1) leave out unused device API's, reducing the size of the compiled application and (2) create your own custom plugin. Every plugin comprises of a piece of JavaScript code and a piece of native code. Calls to the JavaScript API are marshaled and sent to the native counterpart where the request gets unmarshaled and executed. Additionally, a response is be sent back to the caller.

A comprehensive list of Cordova plugins is available at `https://github.com/phonegap/phonegap-plugins`. However, at the time of writing, there is no package manager like `npm`[5] available for Cordova plugins.

4.1.4 Related projects

There are a number of projects related to Apache Cordova that are useful when developing mobile web applications with Cordova.

Adobe PhoneGap Build

Adobe PhoneGap Build[6], part of Adobe Edge Tools & Services, is an online build service for packaging Cordova/PhoneGap applications. Upon request, the application's source code is pulled from a git[7] repository and the packaged app can be downloaded straight from the web interface. An API is available to automate the process. The service supports Android, iOS, Windows Phone, BlackBerry, WebOS and Symbian.

Both free and paid plans (with a monthly subscription fee) are available. The free plan is entitled to only one private project, a paid plan is entitled to 25 or more private apps.

Edge Inspect

Another Adobe product is called Edge Inspect[8], which is also part of Adobe Edge Tools & Services. It comprises of a desktop application, a Chrome plugin and a couple of native applications and allows developers to preview web designs on multiple displays at a time. When the plugin is activated, all connected devices instantly load the same web page. At the same time, developers get access to a WebKit inspector, showing the code of any remote browser (this is based on WEINRE, see further). The application also allows the developer to take screen shots the web pages displayed in a particular browser.

[3] `http://dev.w3.org/html5/webdatabase/`
[4] `http://dev.w3.org/html5/webstorage/`
[5] NPM is the Node Package Manager, `http://npmjs.org`
[6] `http://build.phonegap.com`
[7] Git is a popular version control system, `http://git-scm.org`
[8] `http://html.adobe.com/edge/inspect/`

WEINRE

WEINRE[9] is part of Apache Cordova and is an acronym for "WEb INspector REmote". It provides access to a fully functional WebKit inspector of remote browsers. The service runs on a node[10] server. The only requirement is to include a javascript file in the application's HTML. When the browser starts executing the JavaScript file, it makes a persistent connection with a server through a web socket. This connection is then used for all communication between the web application and the WebKit inspector. This allows developers to inspect the DOM, resources, run custom JavaScript commands, etc. remotely.

A hosted version is available at `http://debug.phonegap.com`.

Apache Ripple

Apache Ripple[11] is a web-based mobile environment simulator and is available as a Chrome extension. It exposes the Apache Cordova device API's and allows developers to simulate various device features in a desktop browser. For instance, you can shake the device, set a location and heading, emulate disruptive networking, etc.

4.1.5 Proof-of-concept application implementation

This subsection provides an overview and short description of the technologies that were used to implement the proof-of-concept application.

The proof-of concept application is implemented as a single-page application. This eliminates page loads and thus improves overall performance of the application. The application code is all wrapped inside the native shell. Instead, future versions could serve application code from a remote server and use AppCache to cache the application data on the device. This way, the application can receive updates without the need of updating the outer shell. The application logic is built with AngularJS, the user interface is built on top of Bootstrap 3. The development workflow is based on Yeoman, which provides useful tools for scaffolding, building, previewing, testing, etc. Unit and integration tests are created with the Jasmine test framework and run with the Karma test runner.

AngularJS

AngularJS[12] is an JavaScript MVC application framework, developed at Google. With AngularJS, developers can use a declarative programming style rather than an imperative style. This is achieved through additional HTML elements and attributes, which serve as annotations. In addition, the imperative programming style is still available to program controller code. AngularJS also provides a dependency injection system which makes

[9] `http://people.apache.org/~pmuellr/weinre/docs/latest/`
[10] Server-side JavaScript, `http://nodejs.org`
[11] `http://ripple.incubator.apache.org`
[12] `http://angularjs.org`

testing so much easier. Components that require user interaction or are not useful in a testing environment can easily be replaced with a mock.

AngularJS has proven useful from the start. The project came to live at Google because developers of the Google Feedback project were unsatisfied with the development speed. The project that already took 6 months to create and 1700 lines of traditional HTML and JavaScript could be completely rewritten in three weeks by a single individual and with only 1500 lines of code [34].

Bootstrap 3

Bootstrap[13] is a popular html user interface framework. Version 3 is a complete overhaul of the project, introducing a mobile-first approach. Instead of scaling down desktop pages, the new version starts from the mobile interface and scales up as the screen size increases.

Yeoman

Yeoman[14] is a collection of tools and best practices to make web development easier. It is comprised of the following tools:

- **Yo** This command-line tool is used for scaffolding application code. A Yo plugin for AngularJS projects is available at `https://github.com/yeoman/generator-angular` and can generate various application artifacts.
- **Grunt** This command-line tool is similar to make[15] and runs various predefined JavaScript tasks like compiling LESS files, minifying javascript and CSS, linting Javascript, running unit tests, etc.
- **Bower** This command-line tool is a package manager for web applications and takes care of hard to track dependencies in complex projects.

Jasmine & Karma

One of AngularJS' focal points is writing testable code. The community has created a test runner, called Karma[16]. Running Karma will start a basic server, open a collection of browser windows (even mobile browser on connected devices or a headless browser like PhantomJS[17]) and run all the tests in these browsers. The results of the tests are then passed back to the test runner in order to report them to the user.

The tests are coded with the Jasmine[18] testing framework.

[13] `http://getbootstrap.com`
[14] `http://yeoman.io`
[15] `https://www.gnu.org/software/make/`
[16] `http://karma-runner.github.io/`
[17] `http://phantomjs.org/`
[18] `http://pivotal.github.io/jasmine/`

4.2 Motorola Rhodes

4.2.1 General information & history

Rhodes[19] was developed and released in 2008 by an American company called RhoMobile. Their goal was to enable developers to quickly create native applications for all smartphones. Rhodes applications make use of synchronized local data and take advantage of the device's hardware. With a local synchronization engine, called RhoSync, the emphasis is clearly on data-driven enterprise applications.

In July 2011, Motorola Solutions (the non-mobile, enterprise and government-oriented division, not the division that was acquired by Google) acquired RhoMobile. After the acquisition, Motorola introduced the commercial product RhoElements: a set of specialized features like a barcode scanner, NFC and signature capture. About one year after the acquisition, Motorola removed a lot of the features from the Rhodes framework and included them in RhoElements version 2.

The Rhodes framework is open source and freely available under the MIT license. Based on the statistics of the RhoMobile Google groups page, community activity has significantly decreased in the last year from 722 monthly posts in June 2012 to 102 monthly posts in April 2013. However, based on the contribution history on GitHub, development activity seems to have increased since January 2013.

4.2.2 Supported platforms

Rhodes currently supports all major platforms, including Android (2.1 and up), iOS (3.0 and up), Windows Phone 8 and BlackBerry (4, 5, 6 and 7).

4.2.3 Architecture

Rhodes is a cross-platform tool that creates applications that are both interpreted and hybrid at the same time. A Rhodes application is a mobile web application that runs locally on top of a Ruby web server, i.e. the application code it written in Ruby, the view is written in HTML. The architecture is depicted in Figure 4.2.

Rhodes also has a plugin system that allows developers to create native extensions for missing functionality. A plugin consists of a shared Ruby interface definition and native implementations for the supported platforms.

4.2.4 Related projects

Rhodes is an open-source framework is part of a collection of software, called RhoMobile Suite, which also contains other software products that fit strategically in this suite. These products are discussed briefly in this section.

[19] http://docs.rhomobile.com

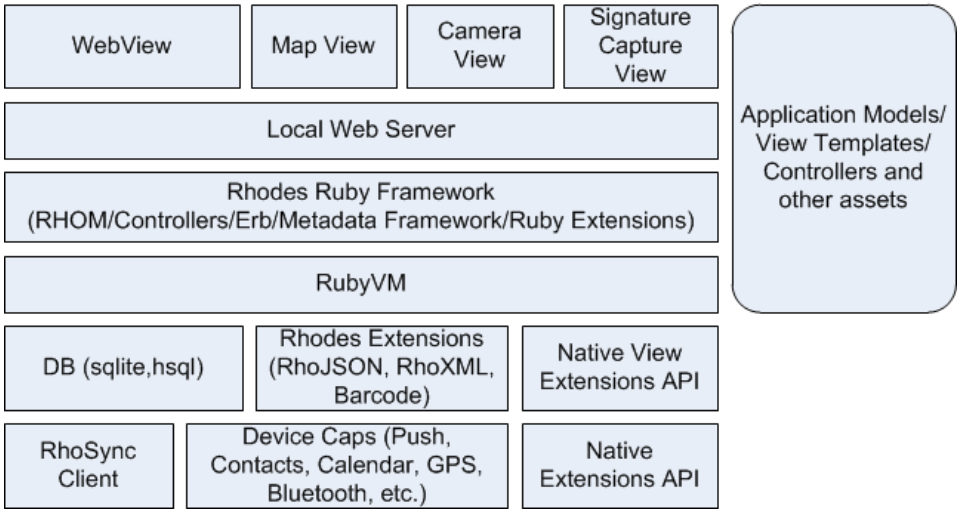

FIGURE 4.2: The architecture of Rhodes applications

RhoElements

RhoElements is a commercial product in the RhoMobile Suite and therefore requires a license. It contains a substantial amount of additional functionality that is useful for enterprise applications like NFC, barcode readers, etc. This functionality was removed from Rhodes in version 3.3.3.

RhoConnect

RhoConnect is another open-source product in the RhoMobile Suite and serves as a mobile orchestrator. It takes care of data synchronization between backend applications and mobile devices. The libraries for this integration are already available in the Rhodes framework.

RhoHub

RhoHub[20] is a cloud platform that developers can use to host and build their applications. Developers can push their code to a private git repository from which the application can be built for Android, iOS, etc.

RhoStudio

RhoStudio is an Ruby IDE, based on Eclipse 3.7, and contains a custom plugin for Rhodes and RhoElements application development. It is freely available as part of the RhoMobile Suite.

[20] https://app.rhohub.com/

4.2.5 Proof-of-concept application implementation

The development of the proof-of-concept application was aborted early because the tooling was bugged and the performance of the resulting Rhodes applications was unsatisfying. On an entry-level smartphone, loading a page took 30 seconds and on a high-end smartphone page loads still take 7 seconds. Capgemini deemed this unacceptable and Rhodes was no longer considered a viable alternative.

Chapter

5 Evaluation

This chapter presents a detailed description of the evaluation of the alternatives, which are the combination of native Android and native iOS, Apache Cordova and Motorola Rhodes. The chapter is structured as outlined in section 3.5.2: first, the weights of the criteria are calculated. Second, the weights of the alternatives are calculated and last, a cost-benefit analysis is performed.

5.1 Weighting the evaluation criteria

Because developers and architects do not necessarily share the same opinions about the evaluation criteria, the ranking of the alternatives could be different as well. Therefore, this evaluation covers these two perspectives separately and the results for both perspectives are compared in the end. Both the developer and the architect were asked to fill in a questionnaire containing all necessary pairwise comparisons (see Appendix **??**).

The questioned people are the same people from the interviews earlier (see section 3.4.1) and the results of the questionnaire are illustrated in Table 5.1, Table 5.2, Table 5.3, Table 5.6, Table 5.4, Table 5.5.

Architect	Portability	App. Experience	Productivity	Priority vector
Portability	1	1/3	1/5	0.1047
App. Experience	3	1	5	0.6370
Productivity	5	1/5	1	0.2583
		$\lambda_{max} = 3.5206$, $CI = 0.2603$, $RI = 0.58$, $CR = 0.4488$		

Developer	Portability	App. Experience	Productivity	Priority vector
Portability	1	1/7	1/5	0.0719
App. Experience	7	1	3	0.6491
Productivity	5	1/3	1	0.2790
		$\lambda_{max} = 3.0649$, $CI = 0.0324$, $RI = 0.58$, $CR = 0.0559$		

TABLE 5.1: Pairwise comparison matrix for the top level of the decision tree.

The weights for the top level criteria are virtually the same for both the architect and the developer perspective. Table 5.2 also reveals a high inconsistency in the architect's judgement ($CR > 0.1$). Analysis of the judgements shows that there is no loop, i.e. there is no chain A, B, C such that A is more important than B, B is more important than C and C is more important than A. However, the judgement of portability vs. application experience does not reflect the judgement of application experience vs. productivity and the judgement of productivity vs. portability since $1/3 \neq 1/5 \times 1/5$. The inconsistency ratio for the judgement of the developer is acceptable as it is below 10%.

TODO: Ask for a better judgement?

Architect	Platform support	Toolset reuse	Code reuse	Priority vector
Platform support	1	1/3	1/7	0.0918
Toolset reuse	3	1	5	0.6248
Code reuse	7	1/5	1	0.2834
			$\lambda_{max} = 3.7089$, $CI = 0.2545$, $RI = 0.58$, $CR = 0.6112$	

Developer	Platform support	Toolset reuse	Code reuse	Priority vector
Platform support	1	1/5	1/5	0.0909…
Toolset reuse	5	1	1	0.4545…
Code reuse	5	1	1	0.4545…
			$\lambda_{max} = 3$, $CI = 0$, $RI = 0.58$, $CR = 0$	

TABLE 5.2: Pairwise comparison matrix for the category "portability".

In the portability category (see Table 5.2), the differences between developer and architect are more pronounced. Just like before, the judgement of the architect shows a notable amount of inconsistency. Again, there is no loop but the comparison data violates the consistency property.

TODO: Ask for better judgement?

TODO: Add a note about statistical insignificance

5.2 Evaluate the alternatives

In the subsections to follow, all alternatives get evaluated with respect to one criterion at a time. Every subsection gives a description of what is measured, how it is measured. Additionally, the description also illustrates what is taken into account for the evaluation, which scale is used and why. For every alternative, a rationale is provided for the judgement or attributed score.

For some criteria, there is no absolute scale available to measure said property. In that case, judgements based on pairwise comparison, as proposed by Saaty **Saaty:1980** are

Architect	Native int.	UI capabilities	Performance	Priority vector
Native int.	1	1	3	0.4600
UI capabilities	1	1	1/3	0.2211
Performance	1/3	3	1	0.3189
			$\lambda_{max} = 3.5608$, $CI = 0.2804$, $RI = 0.58$, $CR = 0.4835$	

Developer	Native int.	UI capabilities	Performance	Priority vector
Native int.	1	1	1/5	0.1428
UI capabilities	1	1	1/5	0.1428
Performance	5	5	1	0.7142
			$\lambda_{max} = 3$, $CI = 0$, $RI = 0.58$, $CR = 0$	

TABLE 5.3: Pairwise comparison matrix for the category "Application Experience".

Architect	Access to HW	Platform-specific svc.	Priority vector
Access to HW	1	1/3	0.2500
Platform-specific svc.	3	1	0.7500
		$\lambda_{max} = 2$, $CI = 0$, $RI = 0$, $CR = 0$	

Developer	Access to HW	Platform-specific svc.	Priority vector
Access to HW	1	5	0.8333...
Platform-specific svc.	1/5	1	0.1666...
		$\lambda_{max} = 2$, $CI = 0$, $RI = 0$, $CR = 0$	

TABLE 5.4: Pairwise comparison matrix for the category "Native integration" within the category "Application Experience".

Architect	Native L& F	UI el. capabilities	Priority vector
Native L& F	1	5	0.8333...
UI el. capabilities	1/5	1	0.1666...
		$\lambda_{max} = 2$, $CI = 0$, $RI = 0$, $CR = 0$	

Developer	Native L& F	UI el. capabilities	Priority vector
Native L& F	1	1/5	0.1666...
UI el. capabilities	5	1	0.8333...
		$\lambda_{max} = 2$, $CI = 0$, $RI = 0$, $CR = 0$	

TABLE 5.5: Pairwise comparison matrix for the category "UI capabilities" within the category "Application Experience".

used to evaluate the alternatives. These judgements arise from the experience of developing the proof-of-concept application. In addition, some criteria deal with features

Architect	Skill reuse	Tooling	Testing	Priority vector
Skill reuse	1	1/5	3	0.2021
Tooling	5	1	5	0.7007
Testing	1/3	1/5	1	0.0972
	$\lambda_{max} = 3.1356$, $CI = 0.0678$, $RI = 0.58$, $CR = 0.1169$			

Developer	Skill reuse	Tooling	Testing	Priority vector
Skill reuse	1	1	3	0.4286
Tooling	1	1	3	0.4286
Testing	1/3	1/3	1	0.1428
	$\lambda_{max} = 3$, $CI = 0$, $RI = 0.58$, $CR = 0$			

TABLE 5.6: Pairwise comparison matrix for the category "Productivity".

that are not addressed in the proof-of-concept application. In that case, the evaluation is based on the information that is found in the documentation.

TODO: Make sure every section clearly mentions these things...

5.2.1 Platform support

Remember from section 3.1 that support for Android and iOS is a selection criterion. This criterion only covers additional platforms, apart from Android and iOS. To reflect the value of each platform, every alternative is awarded the additional market share it can serve. In the end, these scores are normalized such that the sum of scores is equal to 1. The market share numbers are taken from Figure 1.2.

ANDROID & IOS Native code for Android and iOS is useless on other platforms. Hence, the combination of Android and iOS cannot contribute to reaching additional market share.

APACHE CORDOVA From Cordova's platform support (see section 4.1.2), it is safe to say that it supports the remainder of the mobile market. Cordova is therefore awarded the remaining 17% of market share: 6% for BlackBerry, 6% for Symbian, 2% for Windows Phone, 2% for Bada and 1% for the remainder of mobile platforms.

MOTOROLA RHODES Rhodes only supports BlackBerry and Windows Phone for which it is awarded 8% of market share: 6% for BlackBerry and 2% for Windows Phone.

VERDICT The normalized scores are listed in Table 5.7.

5.2.2 Toolset reuse

This criterion describes the ability to reuse the development environment or parts thereof when migrating to another cross-platform alternative. The evaluation is based on the recommended development environment setup for each alternative. Since there is no

Platform support	Market share	Normalized weight
Android/iOS	0%	0
Cordova	17%	0.68
Rhodes	8%	0.32

TABLE 5.7: Evaluation of the alternatives with respect to "platform support".

absolute scale available to measure this property, Saaty's fundamental scale (see Table 2.2) is used here.

ANDROID & IOS All iOS development is done with the Xcode IDE, which is available on Apple computers only. Every iOS developer must therefore have a rather expensive Mac. For Android development, any platform can be used in combination with the Android SDK and an Eclipse plugin (ADT) is available. On I/O 2013, Google also announced a new, freely available Android IDE, based on JetBrains' IntelliJ.

APACHE CORDOVA For HTML5 application development, any operating system with a web IDE or even a simple text editor will do. However, packaging the Cordova application in its native shell requires the native SDK's or the online build service, called PhoneGap Build, provided by Adobe. Unfortunately, this service cannot build additional plugins that are not part of the standard distribution. If a project makes use of custom plugins, the native SDK's are required.

MOTOROLA RHODES Development of Rhodes applications requires a Ruby installation and a Ruby IDE. A specially tailored Eclipse-based Ruby IDE, called RhoStudio, is distributed as part of RhoMobile. As with Cordova, packaging the app requires the native SDK or the online build service, called RhoHub, provided by Motorola.

VERDICT Motorola Rhodes and Apache Cordova have similar requirements and therefore they are treated equally in the comparison. Native development for Android and iOS is treated as a whole. Because of the strict requirements imposed by Apple, native development scores a little less compared to Cordova and Rhodes. However, not being able to build the application locally is not very practical either, which explains the scores listed in Table 5.8.

Toolset reuse	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	1/3	1/3	0.1428
Cordova	3	1	1	0.4286
Rhodes	3	1	1	0.4286
			$\lambda_{max} = 3$, $CI = 0$, $RI = 0.58$, $CR = 0$	

TABLE 5.8: Evaluation of the alternatives with respect to "toolset reuse".

5.2.3 Code reuse

This criterion measures the amount of code that can be reused when migrating to another platform. For this criterion, the amount of portable code is expressed as an estimated percentage of the total amount of application code. Native extensions (through pluggable architectures) are not taken into account as they are not part of the application logic. They are in fact reusable software components that can be coded once and used many times afterwards.

ANDROID & IOS As mentioned before, native code for Android and iOS is useless on other platforms but native Android code is also useless on iOS and vice versa. Hence, code reuse on Android and iOS is 0%.

APACHE CORDOVA Cordova wraps mobile web applications. Hence, all (or 100%) of the application code can be easily shared across platforms. Cordova has a pluggable architecture but — as mentioned in the description — this is not taken into account in this discussion.

Note that a substantial amount of code (or potentially all code) can be reused on other (non-mobile) platforms. Cordova applications only use client-side programming languages and the API's mostly match existing HTML specifications. Be that as it may, it is out of scope for this evaluation and it is not further taken into account either.

MOTOROLA RHODES The application logic of Rhodes applications can be fully shared across platforms as well. Hence, code reuse is also 100%. Rhodes also has plugin system but, these are not taken into account. In contrast to Cordova, Rhodes applications are based on the Rhodes MVC framework which makes it hard (or even impossible) to deploy them outside a Rhodes environment.

VERDICT The scores for each alternative are normalized and listed in Table 5.9.

Toolset reuse	Code reuse	Priority vector
Android/iOS	0%	0
Cordova	100%	0.5000
Rhodes	100%	0.5000

TABLE 5.9: Evaluation of the alternatives with respect to "code reuse".

5.2.4 Access to hardware

This criterion measures the ability to integrate the device's hardware. The availability of device API's (sensor API's, output API's and communication API's) is studied and every alternative is awarded the fraction of device API's it supports out-of-the-box. Alternatives can either provide full support (there is a way to access a particular piece of hardware or software on both platforms), partial support (access to this type of hardware is available on one platform only) or no support. Note that most of these API's are not used in the proof-of-concept application. Therefore, this comparison is mostly based on the documentation of the alternatives.

ANDROID & IOS Android provides support for all types of sensors but that does not imply that all these sensors are available in a particular device. Apple iDevices have less sensors and as such there is no support for every sensor in iOS.

APACHE CORDOVA Cordova tries to promote the web as a first-class platform for mobile applications. "The ultimate purpose of *Cordova* is to cease to exist" [35]. When HTML5 will be fully supported on all mobile browsers, Cordova will no longer be needed. Therefore, Cordova API's mostly match the HTML5 specification and do not provide support for all the sensors and communication because they are simply not in there.

Note that with Cordova's plugin system, it can deliver the same level of integration as native Android and iOS. However, the plugin system is not taken into account here.

MOTOROLA RHODES Rhodes has a unified sensors API which can send events from all types of sensors if they are present. However, Rhodes lacks VideoCapture on all platforms and a MediaPlayer on iOS. Also, Motorola removed quite some useful API's from Rhodes and included them in their commercial product, called RhoElements.

VERDICT The studied device API's and the alternative's support for it are listed in Table 5.10.

API	Android	iOS	Cordova	Rhodes
Accelerometer	✓	✓	✓	✓
Geolocation	✓	✓	✓	✓
Gyroscope	✓	✓	–	✓
Light	✓	✓	–	✓
Magnetic Field	✓	✓	✓	✓
Pressure	✓	–	–	partial
Proximity	✓	✓	–	✓
Relative Humidity	✓	–	–	partial
Temperature	✓	–	–	partial
Microphone	✓	✓	✓	✓
Camera	✓	✓	✓	partial
Hardware buttons	✓	–	partial	partial
Vibration Motor	✓	✓	✓	partial
Speaker	✓	✓	✓	partial
System information	✓	✓	✓	✓
File System	✓	✓	✓	✓
Contacts	✓	✓	✓	✓
Bluetooth	✓	✓	–	✓
NFC	✓	–	–	–
Wi-Fi Direct	✓	–	–	–
USB / Accessory	✓	✓	–	–
Support	Average: 18/21		10.5/21	14.5/21

TABLE 5.10: Evaluation of the alternatives with respect to "access to hardware".

5.2.5 Integration with platform-specific services

This criterion measures the ability of an alternative to integrate the application with platform-specific services like push notifications. The proof-of-concept application does not require this kind of integration and thus this evaluation is based on the documentation. Because there is no absolute scale to quantify this property, the evaluation is also based on judgements.

ANDROID & IOS Platform-specific services are part of the native SDK's. Therefore, this kind of integration is implemented the easiest in the native applications.

APACHE CORDOVA There are no API's for this kind of integration in the default distribution. Fortunately though, Cordova ships with an elaborate plugin-system which can be used to deal with it.

MOTOROLA RHODES In Rhodes, there are no API's for this integration either. However, Rhodes also has a plugin system which can make this integration possible.

VERDICT Integrating with platform-specific services is obviously more easy when programming natively. However, the other alternatives can use their plugin system to provide this integration. Since plugins can be reused easily, this is only deemed to be a minor disadvantage which is visible in the judgements, listed in Table 5.11.

Performance	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	3	3	0.6000
Cordova	1/3	1	1	0.2000
Rhodes	1/3	1	1	0.2000
			$\lambda_{max} = 3$, $CI = 0$, $RI = 0.58$, $CR = 0$	

TABLE 5.11: Evaluation of the alternatives with respect to "platform-specific services"

5.2.6 Native Look & Feel

This criterion indicates whether or not an alternative is able to use native user interface elements. An application can either support the use of native user interface elements fully, partially or not at all.

ANDROID & IOS It needs no explaining that Android and iOS can use the native user interface elements.

APACHE CORDOVA The user interface of Cordova applications is built with HTML5, CSS and JavaScript and there is no possibility to use native user interface elements for it.

However, numerous attempts are made to recreate the native look & feel on top of HTML (like KendoUI, Moobile, etc.). Unfortunately, these projects need a lot of additional code to mimic the behaviour of native interface elements which influences the overall performance of the application in a adverse way.

MOTOROLA RHODES The user interface of Rhodes applications is also built with HTML and consequently cannot deliver the native look & feel to the end user.

VERDICT Despite the efforts of many projects to mock the native look & feel in web applications, it remains a privilege of Android and iOS. Hence, Android and iOS clearly do better compared to Cordova and Rhodes, as is illustrated in Table 5.12.

Native Look & Feel	Native L&F	Priority vector
Android/iOS	Yes	1
Cordova	No	0
Rhodes	No	

TABLE 5.12: Evaluation of the alternatives with respect to "Native look & feel".

5.2.7 UI element capabilities

This criterion measures an alternative's potential to present a rich user interface. Both Cordova and Rhodes rely on HTML for their user interface. Therefore, the evaluation is based on a comparison of the native UI elements with their HTML5 counterparts. Plain HTML only provides basic interface elements, which is why an additional UI framework, called Bootstrap[1], was used to develop the proof-of-concept application.

ANDROID & IOS Android and iOS both have an elaborate portfolio of rich user interface elements. iOS lacks some basic form elements like radio buttons and checkboxes but these can be easily replaced with pickers and switches respectively.

APACHE CORDOVA Cordova applications are mobile web applications that are wrapped in a single webview. Hence, only HTML5 can be used to create the user interface. An application could be comprised of multiple pages or it could be comprised of a single page, in which case it is called a single-page application. The former approach requires page reloads, which feels unnatural in a mobile app (compared to native apps) whereas the latter approach can make the hybrid app behave more like a native app.

The Cordova proof-of-concept application is implemented as a single-page application with AngularJS[2]. This framework drastically improves productivity by reducing the required lines of code through declarative programming and bidirectional bindings between view and model.

MOTOROLA RHODES Rhodes also relies on HTML for its user interface. In contrast to Cordova, Rhodes is actually a complete web server with an MVC framework on top. Every request is handled locally by a controller action and the result is served in a webview. Developers can still choose to create single-page applications but device integration has to be realized through AJAX[3] calls, which requires extra work. The proof-of-concept application is implemented as a multi-page application because this is the default way to develop applications with Rhodes.

VERDICT The scores are listed in Table 5.13.

[1] http://getbootstrap.com
[2] http://angularjs.org
[3] Asynchronous JavaScript and XML

Android	iOS	HTML5	Bootstrap 3
ActionBar	UINavigationBar	–	`navbar` class
Split ActionBar	UIToolbar	–	`navbar` class
Tabs	UITabBar	–	`nav-tabs` class
ListView	UITableView	–	`list-group` class
GridList	UICollectionView	–	Grid system
ScrollView	UIScrollView	by default	–
Spinner	–	`<select></select>`	`dropdown` class
Picker	UIPickerView	`<select></select>`	`modal` class
DatePicker	UIDatePicker	`<input type="date">`	–
Button	UIButton	`<button></button>`	`btn` class
Text field	UITextField	`<input type="...">`	–
Slider	UISlider	`<input type="range">`	–
Progress bar	UIProgressBar	`<progress>`	`progress` class
Options Menu	UIActionSheet	–	–
Checkbox	–	`<input type="checkbox">`	–
Radio	–	`<input type="radio">`	–
On/off switch	UISwitch	–	`btn-group` class
Dialog	UIDialog	–	`modal` class
Alert	UIAlertView	`alert(string)`	–
Popup	–	–	`modal` class
Toast	–	–	`alert` class
TextView	UILabel	any text element	–
MapView	MKMapView	–	–
–	Popover (iPad)	–	`popover` script
Master-Detail	SplitView (iPad)	–	Grid System
WebView	UIWebView	`<iframe>`	–
Android & iOS combined: 26/26		HTML5 & Bootstrap combined: 24/26	

TABLE 5.13: Evaluation of the alternatives with respect to “UI element capabilities”.

5.2.8 Performance

The performance criterion measures the responsiveness of the user interface and the application in general. Responsiveness in this case does not mean the ability to reflow UI elements to fit other screen sizes, responsiveness in this case represents the time it takes an application to respond to a trigger, like pushing a button.

Comparing the performance of these alternatives is hard because their architectures are completely different. For web applications, page load times could be a viable metric. However, only the Rhodes application uses page loads. There is no notion of page loads in native applications and the Cordova application is implemented as a single-page application. Hence, no absolute scale is available and the alternatives are evaluated based on judgement. The analytic hierarchy process can deal with inconsistency as long as

the consistency ratio is low, i.e equal or below 10%. In that case, the judgements are accepted.

Every implementation of the proof-of-concept application is tested on four devices:

- **HTC Wildfire S** An entry-level smartphone, released in May 2011 and running Android 2.3.5 on a single core 600 MHz ARMv6 processor. It has 512 MB of RAM and a 3.2-inch display with a 320 by 480 pixel resolution (180 ppi).
- **HTC Flyer** An entry-level tablet, also released in May 2011 and running Android 3.2.2 on a single core 1.5 GHz ARMv7 processor. It has 1024 MB of RAM and a 7.0-inch display with a 600 by 1024 pixel resolution (170 ppi).
- **LG/Google Nexus 4** A high-end smartphone, released in November 2012 and running Android 4.2.2 on a quad core 1.5 GHz ARMv7 processor. It has 2048 MB of RAM and a 4.7-inch display with a 768 by 1280 pixel resolution (318 ppi).
- **iPhone 3GS** A smartphone, released in June 2009 and running iOS 6.1.3 on a single core 600 MHz ARMv6 processor. It has 256 MB of RAM and a 3.5-inch display with a 320 by 480 pixel resolution (165 ppi).

ANDROID & IOS The native SDK's are the most suited to take advantage of the device capabilities. Therefore, native applications always perform the best, provided they are well-designed.

APACHE CORDOVA Cordova wraps web applications. Hence, the performance of its applications greatly depends on the performance of the browser on the device. On older smartphones, the performance of the browser is often seriously lacking. The use of a single-page application solves this issue in a way, but the performance drop is still notable.

MOTOROLA RHODES A Rhodes application also relies on the native browser to display its user interface. The difference with Cordova is that Rhodes uses a local ruby web server to serve multi-page applications. The performance of the local web server is absolutely disappointing, resulting in utterly slow page loads (of up to 30 seconds for a single load on the slowest devices and 7 seconds on the fastest device).

VERDICT The judgements for this evaluation are listed in Table 5.14. The consistency ratio is rather high but still acceptable.

5.2.9 Skill reuse

This criterion expresses the ability to reuse skills from another field of expertise to develop mobile applications. This allows for better resource management. For this criterion, it is assumed that the organization has got a substantial expertise with Java EE (because Capgemini does).

Performance	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	5	9	0.7352
Cordova	1/5	1	5	0.2067
Rhodes	1/9	1/5	1	0.0581
			$\lambda_{max} = 3$, $CI = 0.0585$, $RI = 0.58$, $CR = 0.10$	

TABLE 5.14: Evaluation of the alternatives with respect to "Performance".

ANDROID & IOS For organizations that already have got expertise with Java EE applications, the entry barrier for adding Android to the portfolio is rather low, since Android is based on Java. Adding iOS from that perspective is less straightforward, because Objective-C is a new programming language.

APACHE CORDOVA In order to develop Cordova applications, one needs to know the web. Since the web already belongs to the organization's expertise, this skill can be reused quite easily but developers will probably have to invest in their JavaScript skills. When custom plugins are needed, the scenario of native Android and iOS holds.

MOTOROLA RHODES For Rhodes applications, organizations can reuse their skills for creating the user interface. However, the application logic entirely written in Ruby, which is a new programming language for the organization.

VERDICT The native SDK's and Rhodes are similar because one needs to learn a new language for either of both. For the development of Cordova application, no notable new skills are required, explaining the scores listed in Table 5.15.

TODO: On the other hand, learning Objective-C appeared to be easier than learning Ruby (in combination with Rhodes) when developing the proof-of-concept application. The free online video lectures about iOS development by Paul Hegarty are very valuable.

Skill reuse	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	1/5	1/5	0.1428
Cordova	5	1	5	0.7144
Rhodes	1	1/5	1	0.1428
			$\lambda_{max} = 3$, $CI = 0$, $RI = 0.58$, $CR = 0$	

TABLE 5.15: Evaluation of the alternatives with respect to "Skill reuse".

5.2.10 Tooling

This criterion describes the overall quality of the supplied tools. This includes command-line interfaces, IDE's, emulators and others. Since there is no absolute scale available to measure the quality accurately, the alternatives are compared in pairs. Saaty's scale is used to express the scores that are based on the experience of developing the proof-of-concept application.

ANDROID & IOS The restrictions on Apple development may seem obstructive at first but it also allows Apple to deliver a well integrated development environment. Everything a developer could possibly need is available in Xcode, right out of the box. For Android development, an Eclipse plugin is available which integrates all the functionality of the SDK with the IDE.

APACHE CORDOVA Cordova does not ship with an elaborate IDE or special development tools. However, there are some tools that can ease the development of mobile web applications (WEINRE, Ripple emulator, etc.). Unfortunately, there is no integrated development environment.

MOTOROLA RHODES Rhodes comes with a bunch of command-line scripts. Sadly, the Android Debug Bridge (ADB) is poorly integrated TODO: (see section **??**) and there is no option to transfer iOS applications to an iDevice from the command-line interface. Instead, either Xcode, iTunes or Fruitstrap is needed for the transfer.

TODO: What about RhoStudio?

VERDICT The scores about the quality of the tools are based on the experience from developing the proof-of-concept application and are listed in Table 5.16.

Tooling	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	3	9	0.6716
Cordova	1/3	1	5	0.2654
Rhodes	1/9	1/5	1	0.0629
		$\lambda_{max} = 3$, $CI = 0.0145$, $RI = 0.58$, $CR = 0.025$		

TABLE 5.16: Evaluation of the alternatives with respect to “Tooling”.

5.2.11 Testing

This criterion measures the ability to (1) write tests and (2) automate testing in a continuous integration environment. This criterion focusses mainly on unit testing and integration testing as these tests are most common. Mobile interface testing is rather hard and is not taken into account for this criterion. Alternatives are evaluated using judgements because there is no absolute scale on which this property could be quantified.

Continuous integration is actually not part of the scope for the proof-of-concept application. Nevertheless, this is an important topic in a production environment and therefore it is examined superficially.

ANDROID & IOS Android and iOS have built-in support for writing unit tests (1). Android is based on Java and consequently, the well-known JUnit framework can be used for writing tests. For iOS, a similar framework is readily available in Xcode for Objective-C.

For automated testing in a continuous integration environment (2), there are a number of options available. Continuous integration testing for Android can be set up quite easily

with Jenkins[4], even headless[5]. For iOS, there are a number of guides available describing how to set it up. However, a Apple hardware is still required. Other options include a hosted continuous integration service.

APACHE CORDOVA For JavaScript, the Jasmine testing framework[6] can be used to write tests (1). In combination with a test runner like Karma[7], these test can be ran in a continuous integration environment. Better yet, this framework allows to automate testing on the actual mobile devices.

MOTOROLA RHODES Unit testing is also built-in into Rhodes. However, only few options seem to be available for automating testing in a continuous integration environment.

VERDICT The judgements are summarized in Table 5.17. All alternatives have support for unit testing. In that perspective, they are all equal. Rhodes seems to be less suited for continuous integration when compared to the other alternatives. The other alternatives perform equally because seem to support the same features.

Testing	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	1	3	0.4286
Cordova	1	1	3	0.4286
Rhodes	1/3	1/3	1	0.1428
			$\lambda_{max} = 3, CI = 0, RI = 0.58, CR = 0$	

TABLE 5.17: Evaluation of the alternatives with respect to "Testing".

Global verdict

With the information from the evaluation above, the total weight a_j of every alternative A_j is calculated as

$$a_j = \sum_{i=0}^{m} c_i w_i \quad j = 1, \ldots, n$$

where c_i is the weight of the criterion C_i and w_i is the weight of alternative A_j with respect to criterion C_i. The total weights are calculated for both the architect and developer and are listed in Table 5.18 and Table 5.19 respectively.

The ranking of the alternatives is the same for both. Conversely, the average total weight can be calculated without changing the order. Android and iOS clearly offer the most benefits with an average total weight of 60.80%. Cordova comes second with an average total weight of 27.04%. Rhodes finishes last with an average total weight of 12.15%.

[4] Jenkins is a popular continuous integration server, http://jenkins-ci.org
[5] On a server without a user interface
[6] http://pivotal.github.io/jasmine/
[7] http://karma-runner.github.io/0.8/index.html

Architect	c_i	$c_i w_{Android/iOS}$	$c_i w_{Cordova}$	$c_i w_{Rhodes}$
Platform support	0.0096	0.0000	0.0065	0.0031
Toolset reuse	0.0654	0.0093	0.0280	0.0280
Code reuse	0.0297	0.0000	0.0148	0.0148
Access to hardware	0.0733	0.0300	0.0175	0.0258
Platform-specific services	0.2198	0.1831	0.0366	0.0000
Native Look & Feel	0.1174	0.1174	0.0000	0.0000
UI element capabilities	0.0235	0.0110	0.0062	0.0062
Performance	0.2031	0.1493	0.0420	0.0118
Skill reuse	0.0522	0.0135	0.0333	0.0055
Tooling	0.1810	0.1216	0.0480	0.0114
Testing	0.0251	0.0108	0.0108	0.0036
a_j		0.6460	0.2438	0.1328

TABLE 5.18: Summary of the total weight and weighted scores of the alternatives with respect to the architect's weights.

Architect	c_i	$c_i w_{Android/iOS}$	$c_i w_{Cordova}$	$c_i w_{Rhodes}$
Platform support	0.0065	0.0000	0.0044	0.0021
Toolset reuse	0.0327	0.0047	0.0140	0.0140
Code reuse	0.0327	0.0000	0.0163	0.0163
Access to hardware	0.0772	0.0316	0.0184	0.0272
Platform-specific services	0.0155	0.0129	0.0026	0.0000
Native Look & Feel	0.0155	0.0155	0.0000	0.0000
UI element capabilities	0.0772	0.0363	0.0205	0.0205
Performance	0.4636	0.3408	0.0958	0.0269
Skill reuse	0.1196	0.0309	0.0762	0.0125
Tooling	0.1196	0.0803	0.0317	0.0075
Testing	0.0398	0.0171	0.0171	0.0057
a_j		0.5700	0.2971	0.1328

TABLE 5.19: Summary of the total weight and weighted scores of the alternatives with respect to the developer's weights.

When comparing the alternatives in pairs and using Saaty's scale (see Table 2.2), one can conclude that:

- Android and iOS are a little better than Cordova (60.8/27.04 = 2.25),
- Android and iOS are a little better than Cordova (60.8/12.15 = 5) and
- Cordova is a little better than Rhodes (27.04/12.15 = 2.22).

5.3 Costs versus benefits

Having the most benefits does not imply that a tool is the most cost-effective. No company should waste their resources on a cross-platform tool that is not profitable. Benefits and costs are deliberately separated from the beginning to allow the decision maker to make a cost-benefit analysis. In this section, the benefits of each alternative will be compared to their costs.

There are two types of costs. The first category, called fixed costs, are independent of the amount of produced code and includes license and subscription fees, developer tools, etc. Because these costs can be shared across projects, their impact on the total cost of a project is rather small. The second category, called variable costs, are dependent of the amount of produced code and includes for instance the rates of the people involved.

Software is essentially a collection of lines of code that is produced in a certain amount of time. This time is expressed in man-months, making abstraction of the amount of people contributing to it, i.e. 2 man-months are equal to the work of 2 individuals during 1 month. Consequently, duration is a good estimate for the total cost of a software project.

Developing natively for both Android and iOS requires about twice the time that is needed to develop the application for one platform only. This is a consequence of the impossibility to reuse code across platforms (see section 5.2.3). Developing mobile web applications for Android and iOS with Apache Cordova requires about half of the development time because 100% of the code can be reused across platforms. The downside of this approach is that about half of the benefits are lost as well. Nevertheless, in terms of cost-effectiveness, both perform equally well. Compared to Cordova, Rhodes applications are estimated to need a little bit more work because it lacks some common but frequently used functionality (like session management). However, the benefits associated with Rhodes are considerably lower than with Cordova. These findings are visualized in Figure 5.1.

Note that adding an additional platform to the native portfolio will increase the development time (and cost) by half, whereas additional platforms can be added at virtually no cost when developing mobile web applications with Cordova. In case a third platform has to be supported, Cordova application will have better value-for-money.

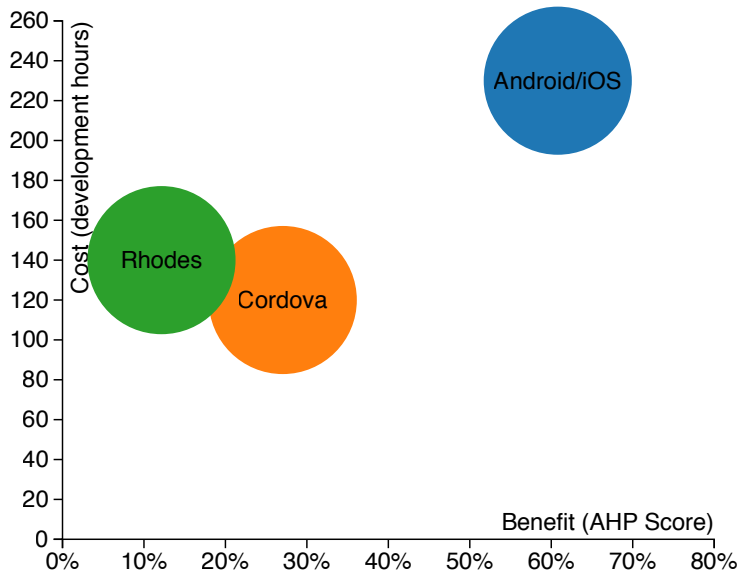

FIGURE 5.1: Visual representation of the cost-benefit analysis.

Chapter

6 Conclusion

This final chapter presents a summary of the work, together with a critical reflection and an overview of potential future improvements.

6.1 Goals

This thesis had two major goals. The first goal was to design a methodology for evaluating and selecting a cross-platform tool for mobile application development. The second goal was to use this methodology to evaluate real cross-platform tools and select the most suited candidate. The methodology that was used is inspired by the generic software package selection, presented by Jadhav and Sonar [30], and comprises of six highly customizable stages. Each of these stages has been concretized in Chapter 3.

During the first stage, the selection criteria were gathered. These are the most essential requirements that a cross-platform tools has to meet. The selection criteria were defined by Capgemini and require cross-platform tools to produce native applications for both Android and iOS and for both smartphones and tablets.

During the second stage, a list of potential candidates was composed from Internet searches and literature. An extensive list of cross-platform tools was found in "Cross-platform developer tools 2012", a report by VisionMobile on this subject [25]. Subsequent Internet searches did not reveal new tools.

During the third stage, this list of potential candidates was filtered using the selection criteria from the first stage and from the resulting list, two cross-platform tools were selected for evaluation. These tools are Apache Cordova and Motorola Rhodes (see Chapter 4). The native development kits for both Android and iOS were also included as a baseline for the evaluation.

During the fourth stage, the evaluation criteria were defined. Three people at Capgemini were interviewed: a developer, a mobile architect and a sales person. From these interviews and literature, eleven evaluation criteria were identified: platform support, toolset reuse, code reuse, access to hardware, integration with platform-specific services, native look & feel, user interface capabilities, performance, skill reuse, tooling and testing.

These criteria are organized in a hierarchy because humans can only process 7 plus or minus 2 pieces of information at the same time.

During the fifth stage, the alternatives were evaluated using these evaluation criteria. For this evaluation, the Analytic Hierarchy Process (AHP) **Saaty:1980** was used. This method assigns weights to both criteria and alternatives based on judgements that originate from pairwise comparisons. In order to formulate a reliable judgement, a proof-of-concept application was implemented with the studied tools. The evaluation is carried out from two perspectives: one from the perspective of a developer, one of a perspective of a mobile architect. Both could have different opinions about cross-platform tool qualities, which could result in a different ranking.

During the sixth and last stage, the candidates that is are most suited were carefully selected. This required a cost-benefit analysis to ascertain that the selected alternative was also cost-effective. For this cost-benefit analysis, development time was used as a cost driver and the scores for the alternatives, obtained from the evaluation, were used as benefits. From this analysis, it was concluded that Apache Cordova is currently equally cost-effective as the native development kits when only targeting Android and iOS. However, if in the future a third platform has to be supported, Cordova will be more cost-effective because the application can be completely reused on the next platform.

6.2 Reflection and future work

The methodology presented in this thesis ends with the selection of a cross-platform tool. However, the usefulness of this tool is never validated. Hence, a seventh, validation stage seems desirable. However, it is not possible to include this step in the timeframe of this thesis because such validation requires that the tool is used in a production environment for quite some time. The prolonged use of a particular tool in a large-scale environment will definitely reveal more benefits and/or issues than a simple proof-of-concept application can in a small-scale and controlled environment. Ergo, this seventh step is of utmost importance after the selection of a cross-platform tool.

Also, the mobile industry is rapidly evolving, which inherently makes cross-platform tools moving targets. The outcome of this study will probably be different in one year from the moment of writing. This makes continuous evaluation of the available tools a necessity and introduces a feedback loop in the evaluation process. New technologies often provide less functionality in the early stages of its life cycle but they can quickly overcome the functionality of the current technologies. New tools can be deemed ineffective at a certain time, but may become more effective than the current tools in the future. Frequent reevaluation is a must.

From the cost-benefit analysis in this study it is concluded that Apache Cordova and the native development kits are currently equally cost-effective. If a company decides to use Cordova for a number of its applications, a new problem rises because Cordova only wraps (mobile) web applications. There is a large number of tools available for (mo-

bile) web development. Working with each of these technologies will create a different experience, both for developers and end-users, which motivates the need for another comparative study, a study that compares tools for (mobile) web development.

During the evaluation phase, the criteria are weighted using the judgements of only two individuals. The evaluation is based on the judgement of the evaluator. Hence, the result does not represent the animo among developers and architects. The result rather represents the combined judgement of three individuals. In future work, it might be wise to increase the sample size of the questioned people and evaluators in order to obtain a statistically valid result. However, having multiple individuals evaluate the alternatives was simply not an option and only two employees were available for questioning.

For the evaluation, the Analytic Hierarchy Process is used. One of the strengths of this method is that it is based on pairwise comparison but this could also become a weakness when a large number alternatives needs to be evaluated. This problem can be solved either by using another evaluation technique or by making modifications to the AHP to deal with these numbers, as is suggested in literature.

Appendices

Appendix

A Requirements documentation of the proof-of-concept application

A.1 Class Diagram

A.2 Mock-ups

A.2.1 Login screen

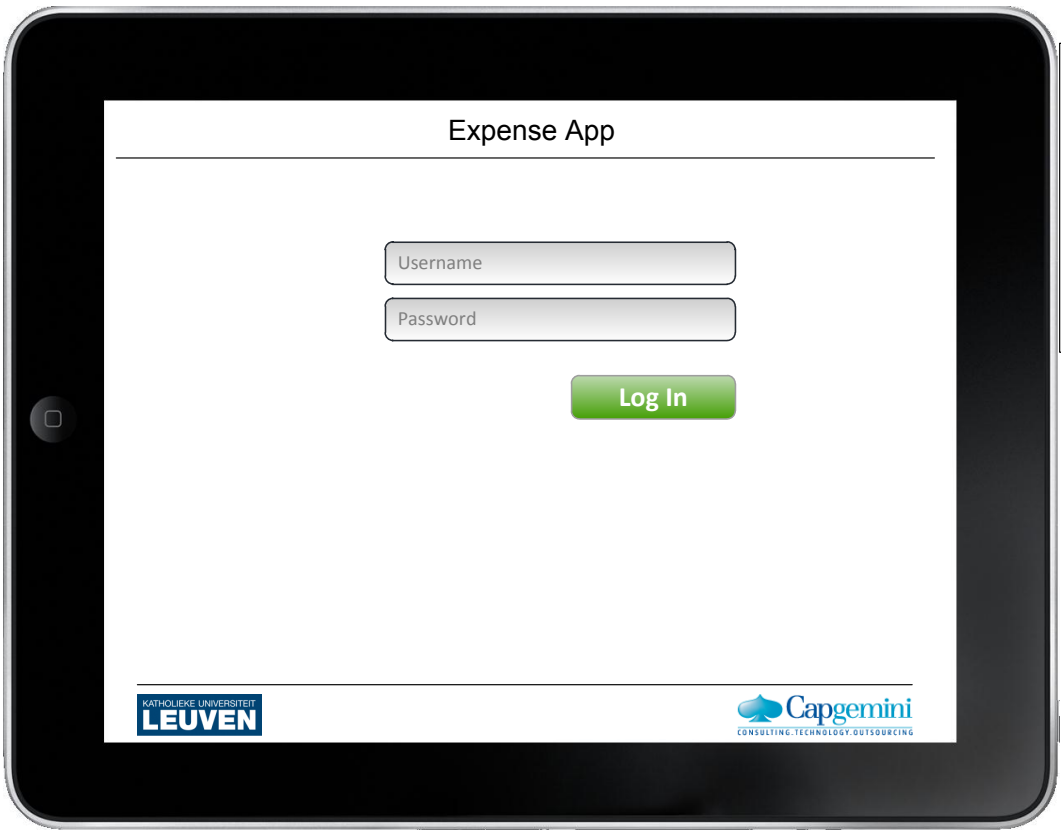

FIGURE A.1: Login screen

Figure A.1 shows the main screen when the user is not logged in. Use credentials to log in.

Validation: both fields must be filled in, otherwise error.

A.2.2 Home screen

A.2.3 Wizard: Step 1 (Your Info)

A.2.4 Wizard: Step 2 (Overview)

A.2.5 Wizard: Step 2 (review expense from abroad)

A.2.6 Wizard: Step 2 (review domestic expense)

A.2.7 Wizard: Step 3 (add expense domestic expense)

A.2.8 Wizard: Step 3 (add expense expense from abroad)

A.2.9 Wizard: Step 4 (Sign & Send)

A.2.10 Wizard: Validation Dialog

A.2.11 Wizard: Validation element coloring

A.2.12 History

A.2.13 Smartphone version

Bibliography

[1] Gartner, *Gartner Says Worldwide Smartphone Sales Grew 16 Per Cent in Second Quarter of 2008*, 2008. [Online]. Available: http://www.gartner.com/it/page.jsp?id=754112.

[2] ——, *Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 11.5 Per Cent Increase in Third Quarter of 2008*, 2008. [Online]. Available: http://www.gartner.com/it/page.jsp?id=827912.

[3] ——, *Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008*, 2009. [Online]. Available: http://www.gartner.com/it/page.jsp?id=910112.

[4] ——, *Gartner Says Worldwide Mobile Phone Sales Grew 17 Per Cent in First Quarter 2010*, 2010. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1372013.

[5] ——, *Gartner Says Worldwide Mobile Device Sales Grew 13.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down*, 2010. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1421013.

[6] ——, *Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent*, 2010. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1466313.

[7] ——, *Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010*, 2011. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1543014.

[8] ——, *Gartner Says 428 Million Mobile Communication Devices Sold Worldwide in First Quarter 2011, a 19 Percent Increase Year-on-Year*, 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1689814.

[9] ——, *Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent*, 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1764714.

[10] ——, *Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent*, 2011. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1848514.

[11] ——, *Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth*, 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1924314.

[12] ——, *Gartner Says Worldwide Sales of Mobile Phones Declined 2 Percent in First Quarter of 2012; Previous Year-over-Year Decline Occurred in Second Quarter of 2009*, 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=2017015.

[13] ——, *Gartner Says Worldwide Sales of Mobile Phones Declined 2.3 Percent in Second Quarter of 2012*, 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=2120015.

[14] ——, *Gartner Says Worldwide Sales of Mobile Phones Declined 3 Percent in Third Quarter of 2012; Smartphone Sales Increased 47 Percent*, 2012. [Online]. Available: http://www.gartner.com/newsroom/id/2237315.

[15] ——, *Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012*, 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2335616.

[16] ——, *Gartner Says Asia/Pacific Led Worldwide Mobile Phone Sales to Growth in First Quarter of 2013*, 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2482816.

[17] Nielsen, *Smartphones Account for Half of all Mobile Phones, Dominate New Phone Purchases in the US*, 2012. [Online]. Available: http://blog.nielsen.com/nielsenwire/online%5C_mobile/smartphones-account-for-half-of-all-mobile-phones-dominate-new-phone-purchases-in-the-us/.

[18] IDC, *Android Expected to Reach Its Peak This Year as Mobile Phone Shipments Slow, According to IDC*, 2012. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23523812.

[19] Gartner, *Gartner Says Apple iOS to Dominate the Media Tablet Market Through 2015, Owning More Than Half of It for the Next Three Years*, 2011. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1626414.

[20] ——, *Gartner Says Worldwide Media Tablets Sales to Reach 119 Million Units in 2012*, 2012. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1980115.

[21] IDC, *IDC Raises Its Worldwide Tablet Forecast on Continued Strong Demand and Forthcoming New Product Launches*, 2012. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23696912.

[22] Android, *Platform Versions, Screen Sizes and Densities and Open GL Version*, 2013. [Online]. Available: http://developer.android.com/about/dashboards/index.html.

[23] D. Smith, *iOS Version Stats*, 2013. [Online]. Available: http://david-smith.org/iosversionstats/.

[24] Chitika, *Post-iOS 6.1.1 Launch, iOS 6 Users Generate 83.1% of all iOS Traffic in North America*, 2013. [Online]. Available: http://chitika.com/ios-version-distribution.

[25] S. Jones, C. Voskoglou, M. Vakulenko, V. Measom, A. Constantinou, and M. Kapetanakis, "Cross-platform developer tools 2012 Bridging the worlds of mobile apps and the web", Tech. Rep., 2012, pp. 1–97. [Online]. Available: http://www.visionmobile.com/product/cross-platform-developer-tools-2012/.

[26] P. Friese, *Cross platform mobile development*, 2012. [Online]. Available: http://www.slideshare.net/peterfriese/cross-platform-mobile-development-11239246.

[27] Apple, *Configuring Web Applications*, 2010. [Online]. Available: http://developer.apple.com/library/safari/%5C#documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html%5C#//apple%5C_ref/doc/uid/TP40002051-CH3-SW4.

[28] M. Mahemoff. (Jun. 2011). HTML5 vs native: the mobile app debate, [Online]. Available: http://www.html5rocks.com/en/mobile/nativedebate/.

[29] A. S. Jadhav and R. M. Sonar, "Evaluating and selecting software packages: A review", *Information and Software Technology*, vol. 51, no. 3, pp. 555–563, Mar. 2009, ISSN: 09505849. DOI: 10.1016/j.infsof.2008.09.003. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0950584908001262.

[30] ——, "Framework for evaluation and selection of the software packages: A hybrid knowledge based system approach", *Journal of Systems and Software*, vol. 84, no. 8, pp. 1394–1407, Aug. 2011, ISSN: 01641212. DOI: 10.1016/j.jss.2011.03.034. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S016412121100077X.

[31] C. Kahraman, *Fuzzy Multi-Criteria Decision Making*, C. Kahraman, Ed., ser. Springer Optimization and Its Applications. Boston, MA: Springer US, 2008, vol. 16, pp. 1–18, ISBN: 978-0-387-76812-0. DOI: 10.1007/978-0-387-76813-7. [Online]. Available: http://www.springerlink.com/index/10.1007/978-0-387-76813-7.

[32] T. L. Saaty, "How to make a decision: The analytic hierarchy process", *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, Sep. 1990, ISSN: 03772217. DOI: 10.1016/0377-2217(90)90057-I. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/037722179090057I.

[33] Gartner, "Cross-Platform Mobile Development Frameworks", Tech. Rep., 2011. [Online]. Available: http://www.gartner.com/id=1848214.

[34] B. Green and S. Seshadri, *AngularJS*. O'Reily, 2013, p. 196, ISBN: 9781449344856.

[35] B. LeRoux, *PhoneGap Beliefs, Goals, and Philosophy*, 2012. [Online]. Available: http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy/.

Fiche masterproef

Student: Michiel Staessen

Titel: A comparative study of cross-platform tools for mobile application development

Nederlandse titel: "Een vergelijkende studie van cross-platform tools voor het ontwikkelen van mobiele applicaties"

UDC:

Korte inhoud:

Developing mobile applications (apps) for multiple platforms is an expensive and time consuming process. Therefore, many companies are seeking refuge in Cross-Platform Tools (CPTs) for the development of their apps. In cooperation with CapGemini, this thesis presents a comparison of two such cross-platform tools: Apache Cordova and Motorola Rhodes. Both tools are compared with each other and with native development. The comparison is based on a proof-of-concept application which should work on both smartphones and tablets with respect to both iOS and Android.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Software engineering

Promotor: prof. dr. ir. Erik Duval

Assessor:

Begeleider: ir. Gonzalo Parra