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

Assessoren:
ir. Gonzalo Parra
prof. dr. Danny Hughes

Begeleider:
ir. Gonzalo Parra

Academiejaar 2012 – 2013

Voorwoord

Contents

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

Mobile phones have been around since the nineties 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 capabilities. 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[2] 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 applications or apps for short, on their device. These application play an important role as they drive the network effects associated with a certain platform. Applications create additional value for a platform, which attracts new users. Whenever a platform attracts more users, it becomes even more valuable. This vicious circle is called a network effect. Network effects make it hard for new platforms to gain traction which is visible in Figure 1.1 and Figure 1.2. Android and iOS

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

[2]this is the ratio between smartphones and all mobile phones

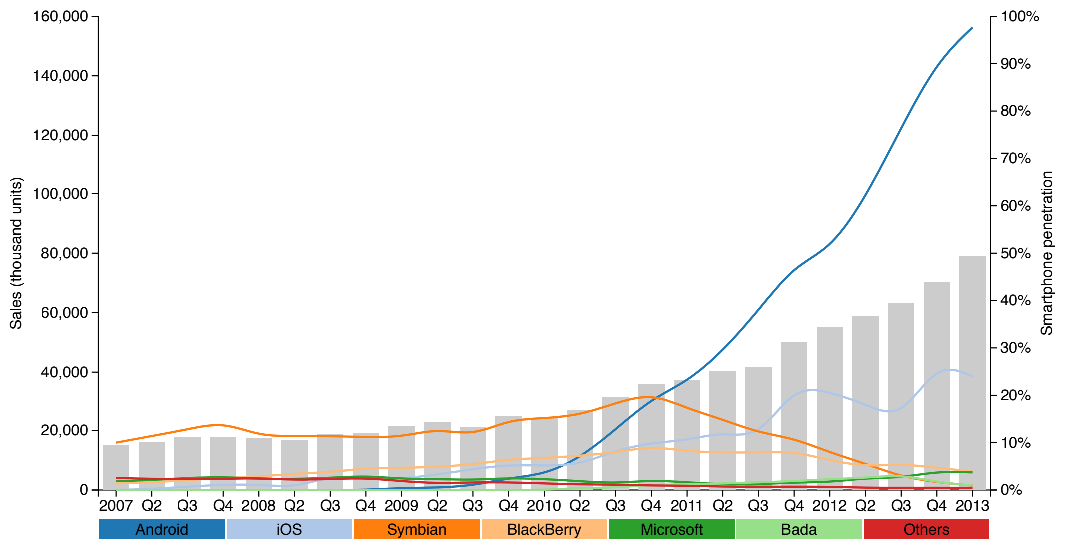

FIGURE 1.1: Evolution of worldwide smartphone sales by operating system (lines) and smartphone penetration (bars). Source: Gartner [1]–[16]

are currently dominant (and continue growing) while 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).

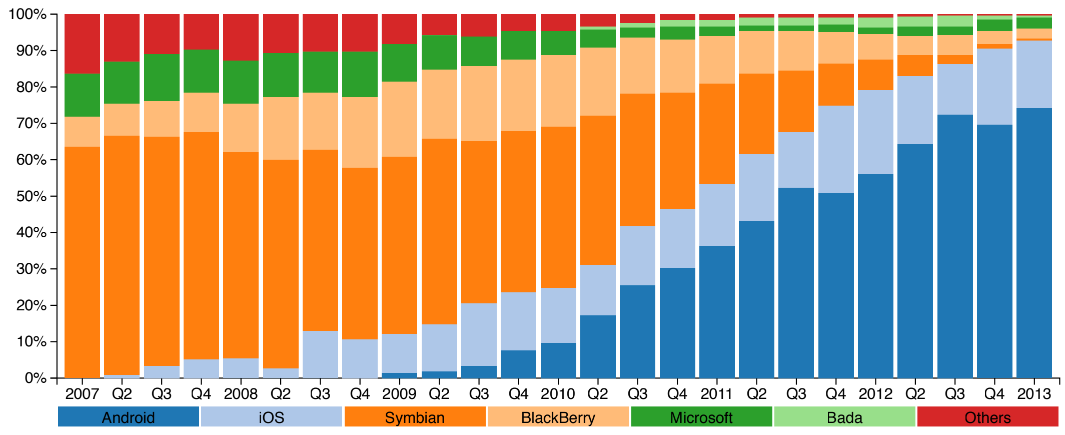

FIGURE 1.2: Evolution of worldwide smartphone market share by operating system. Source: Gartner [1]–[16]

However, there is no single major platform. In addition, the IDC[3] predicts that Windows Phone will gain a significant market share by 2016 and that 90% of the worldwide smart-

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

phone 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.

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!

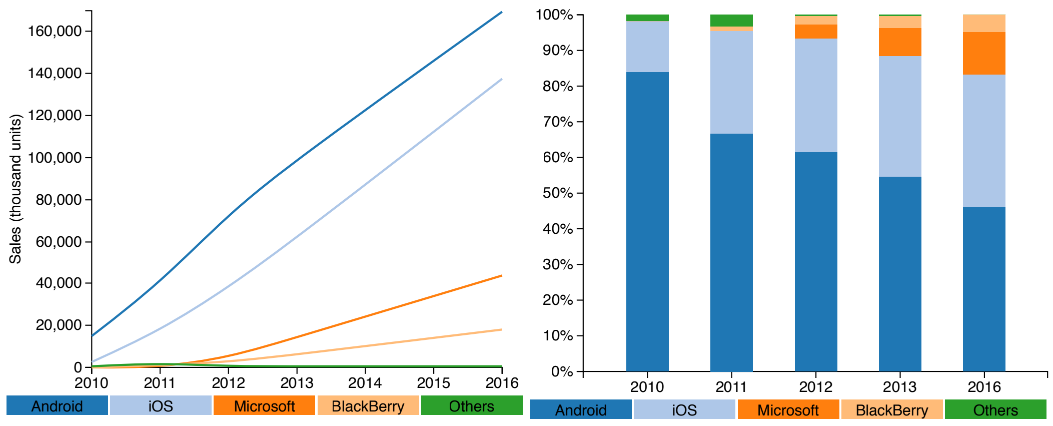

FIGURE 1.3: Prediction of worldwide tablet sales and market share. Source: Gartner [19], [20]

1.2 The problem of fragmentation

Fragmentation can be defined as the fact that all devices are different; there is no uniform device. This is called device 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 problem.

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.

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.

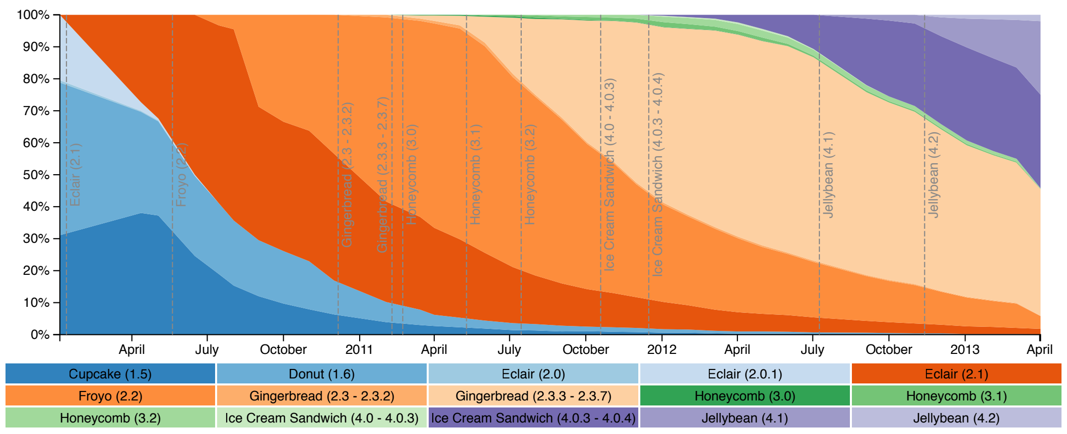

FIGURE 1.4: Historical Android runtime fragmentation. The data for this graph was aggregated from [22] using the Internet Archive (http://archive.org).

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, as Apple starts to drop support for some of its devices that are still in use (like for instance the original iPad and the iPod third generation), it introduces runtime fragmentation. Because of this, iOS developers will have to support the latest two versions of iOS.

Android and iOS are used in both smartphones and tablets. There are only few different iDevices but in contrast, there is an enormous number of Android devices. The Google Play Store[4] supports 2935 different device configurations, distributed across 63 manufacturers. All these devices come with varying screen sizes and resolutions which implies that applications also have to deal with this. 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 of the iPhone 5 (or iPod Touch, 5th generation) will have black bars on the top and the bottom. High-density displays[5], like the Retina display, 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 conclusion, fragmentation among Android devices is rather high compared to iDevices.

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) are built on equipment or software of a single vendor which puts that vendor in a bargaining position. Companies will try to avoid these situations at all costs and they will do so by asking for cross-platform solutions.

Making a solution work across platforms typically requires that the software has to be 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.

[4] Google Play is the official marketplace for Android applications.
[5] A high-density display has about twice the pixel density of a regular display.

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. Bear in mind 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 introduced, 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 executing other maintenance tasks across codebases. This reduces cost while increasing productivity.

1.4 Goals

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 two cross-platform tools (Apache Cordova and Motorola Rhodes) and the native SDK's in a business environment, provided by Capgemini[6]. From this evaluation, the best-suited alternative will be selected.

[6]Capgemini is a multinational IT consulting company, `http://www.capgemini.com`.

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 serves many purposes: 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 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.

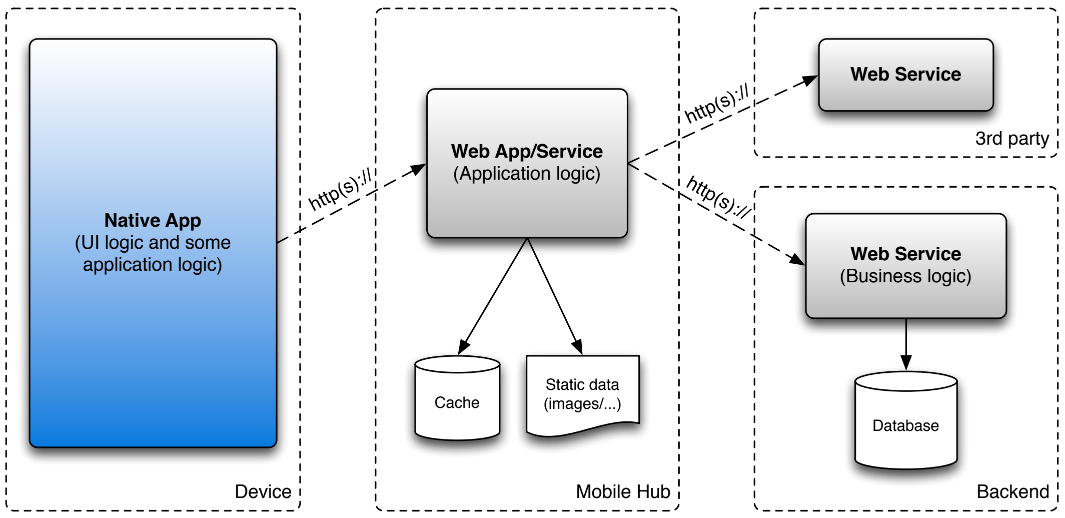

FIGURE 2.1: Architecture of a native application. Based on [26]

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.

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 of the device.

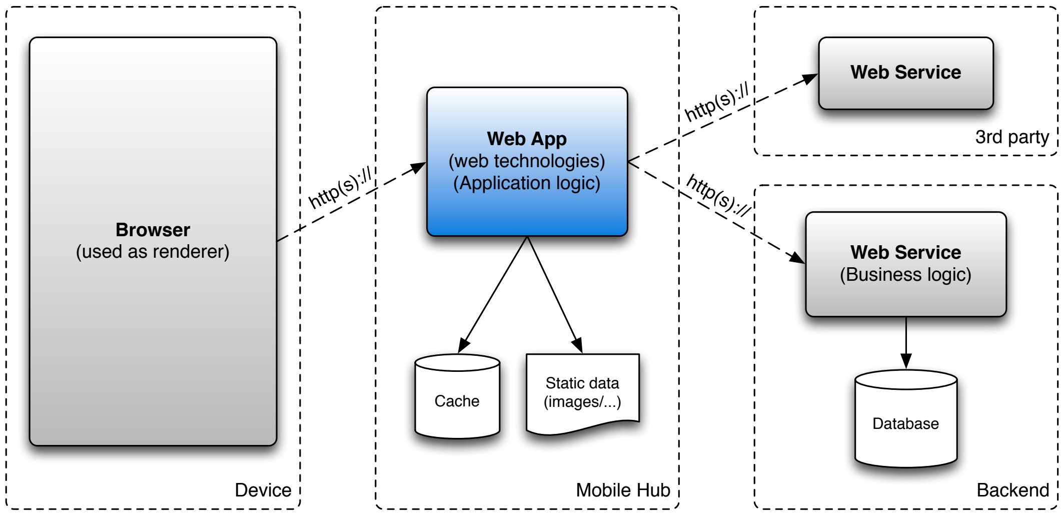

FIGURE 2.2: Architecture of a mobile web application. Based on [26]

Mobile web applications are not nearly as powerful as native applications. First, the application is not stored on the device and requires an active internet connection which is not always available. Second, the performance of the mobile browser is often disappointing. Third, mobile web application cannot make use of the many unique features of mobile devices as the APIs are missing.

A new HTML specification, called HTML5 is here to tackle these issues. It enables mobile web applications to access certain device features and allows applications to be cached entirely on the device. However, the web is currently not a first-class platform and many mobile web browsers lack support for several 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 or not. 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.

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].

[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 are wrapped in a native shell, they can be distributed through (official) marketplaces. The architecture is visualized in Figure 2.3.

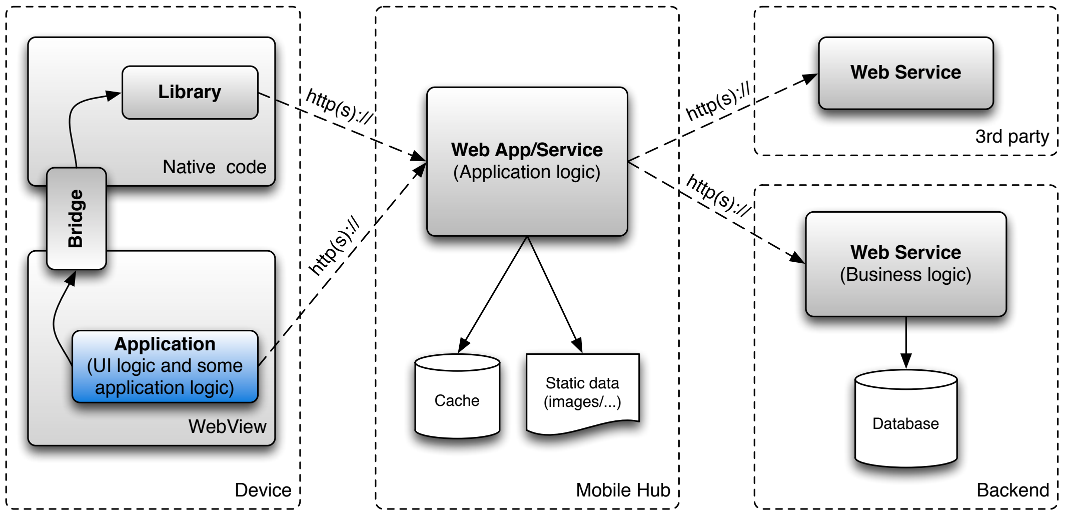

FIGURE 2.3: Architecture of a hybrid application. Based on [26]

Due to the nature of the application, the performance is 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.

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 runtime 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.

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

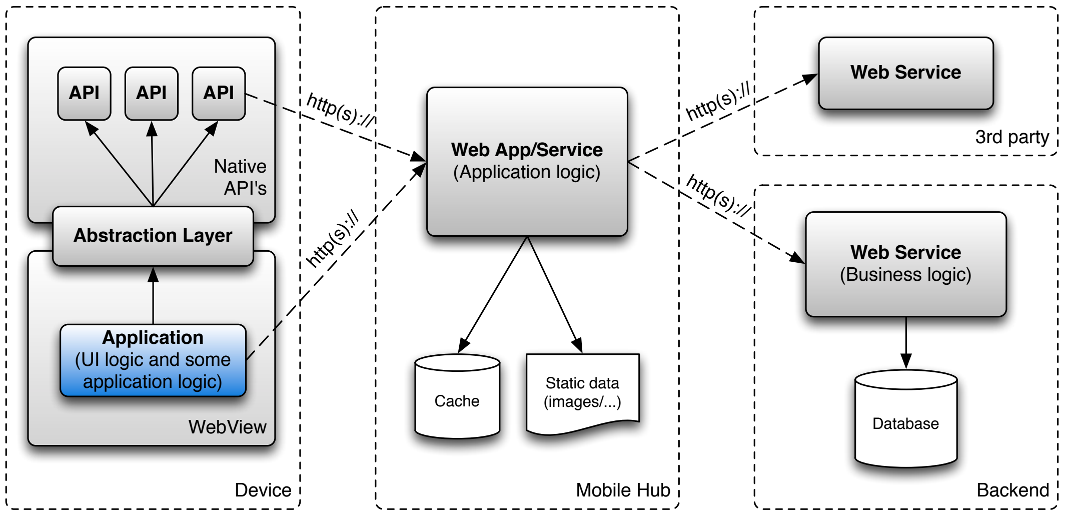

FIGURE 2.4: Architecture of an interpreted or runtime application. Based on [26]

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.

2.1.5 Cross-compiled application

In contrast to interpreted applications, the application logic of cross-compiled is translated into native code 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 is only composed 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.

Comparison of mobile architectures

Finally, the discussed architectures are compared side by side and 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.

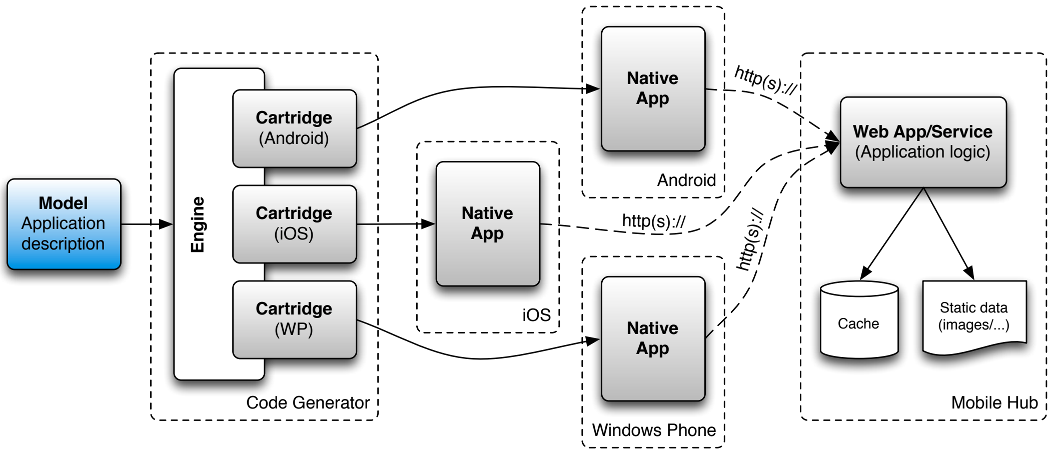

FIGURE 2.5: Architecture of a cross-compiled application. Based on [26]

	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

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

2.2 Software evaluation methodology

One of the goals of this thesis is to design a methodology for evaluating and selecting cross-platform tools. Based on their literature review [29], Jadhav and Sonar [30] propose a generic software selection methodology. This methodology comprises of six steps:

1. **Define selection criteria.** In the first stage, the selection criteria are defined. These are the functional and non-functional requirements that a software package has to meet.

2. **Identify potential candidates.** During the second stage, a list of potential candidates is composed. During this search, any package that could be used to solve the targeted problem is considered a potential candidate.

3. **List selected alternatives.** In the third step, the selection criteria from the first stage are used to filter the list of potential candidates from the second stage: if a candidate does not satisfy the selection criteria, it is not considered for further evaluation.

4. **Define evaluation criteria.** In the fourth stage, the evaluation criteria are defined. These are criteria that will be used to score a particular software package. These criteria should be arranged in a tree and every leaf node should be well-defined and measurable basic attribute.

5. **Evaluate selected alternatives.** During the fifth phase, the software packages are evaluated. First, weights are assigned to the evaluation criteria from the fourth stage. Second, the software packages are rated with respect to the evaluation criteria. Third, an aggregated score is calculated for each software package.

6. **Select the most suitable alternative.** In the final stage, all alternatives are ranked in order of decreasing aggregated score (which was calculated in the previous stage). The most suited software package will rank highest. However, selection is always human-dependable: additional cost-benefit analysis could be performed to identify the package with the best value, contract negotiations can influence the decision, etc.

In their first paper, Jadhav and Sonar [29] also suggest to include a validation stage to confirm that the selected package is indeed the most suited.

2.3 Multicriteria decision making (MCDM)

The generic software selection methodology of Jadhav and Sonar [30] does not impose a particular technique for evaluating software packages. However, the last three stages of this methodology can be recognized as a multi-criteria decision making (MCDM) problem, for which a number of techniques are available. Multi-criteria decision making refers to making decisions in the presence of multiple — often conflicting — criteria.

There are two types of MCDM problems. In multi-attribute decision making (MADM) problems, the decision maker tries to *identify* the "best" alternative from a *finite* set of alternatives with respect to multiple criteria or *attributes*. In multi-objective decision making (MODM) problems, the decision maker tries to *design* the "best" alternative from an *infinite* set of possibilities with respect to multiple *objectives* [31]. The software selection process clearly belongs to the category of MADM problems because the goal is to find the best alternative within a finite set of alternatives [29], [30].

There are numerous approaches to solve MADM problems. In the next subsections, a collection of approaches that are successfully applied in literature will be discussed. Every subsection will present a (high-level) description of the method and its strengths and weaknesses. Subsequently, every method is applied to a simple, running example.

The running example is as follows. Ethan has just graduated and has received three job offers. Every job is evaluated with respect to four criteria: the starting salary, distance to work (in km), degree of interest and career opportunities. Job 1 is an interesting position at a local start-up but it pays less. A large multinational offers job 2, which has a high reward and great career opportunities but is also further away from home. Job 3 is far away from home but combines great career opportunities with a good salary and interesting work. Note that the first two criteria are quantitative criteria, i.e. they can easily be mapped on a number. The last two criteria are qualitative criteria, i.e. they give a description but there is no straightforward mapping on a number.

2.3.1 Scoring models

The first approach to solving MCDM problems is a collection of methods that have one thing in common: every alternative is assigned an arbitrary score with respect to one criterion and the scores for all criteria are aggregated into one final score. The final scores are then used to rank the alternatives.

Different methods are available to calculate the final score of a candidate [31]:

- The "dominance" method can be used when one alternative clearly outperforms the other alternatives with respect to at least one criterion and performs equally well with respect to the other criteria.
- The "maximin" method calculates the total score of an alternative as the lowest score of all criteria scores of this alternative.
- The "maximax" method calculates the final score of an alternative as the highest score of all criteria scores of this alternative.
- With the "conjunctive" method, an alternative should exceed certain thresholds for *all* criteria.
- With the "disjunctive" method, an alternative should exceed certain thresholds for *at least one* criterion.
- The "additive weighting" method calculates the total score of an alternative as the weighted *sum* of the criteria scores and criteria weights.
- The "weighted product" method calculates the total score of an alternative as the weighted *product* of the criteria scores and criteria weights.

The strength of these methods is that they are the easy to use. The weaknesses are that it (1) is hard to assign scores to qualitative criteria and definitely when there are many alternatives because humans can only deal with 7, plus or minus 2 pieces of information simultaneously and (2) different scales get mixed up [29], [32]. These weaknesses will become clear in the example.

EXAMPLE Assume that Ethan has assigned weights to the four criteria in an arbitrary fashion and that he has also assigned scores to every alternative. Because this method requires a numerical value for each criterion, Ethan has created a value scale for "interest" and "opportunities": 0 represents satisfiable, 1 represents good and 2 represents excellent. Also, the distance property has been inverted such that a shorter distance is rewarded with a higher score. He has naively used the additive weighting method to aggregate the scores. The weights, scores, and resulting total scores are listed in Table 2.2.

Alternative	Salary	Distance (km)	Interest	Opportunities	Total Score
Job 1	2000	1/5	2	0	400.62
Job 2	2800	1/30	0	2	560.80
Job 3	2600	1/60	1	1	520.70
Weight	20%	10%	30%	40%	

TABLE 2.2: Naive demonstration of the additive weighting method.

From this first and naive attempt, Ethan would be inclined to choose the second job offer over the third one, and the third offer over the first one. However, there are a number of issues with this approach. The scores have different orders of magnitude, which affects the scaling. This can be solved by normalizing the scores, i.e. dividing each score by the sum of scores for a particular criterion. The new, normalized scores are listed in Table 2.3.

Alternative	Salary	Distance (km)	Interest	Opportunities	Total Score
Job 1	0.27	0.80	0.67	0	0.33
Job 2	0.38	0.13	0	0.67	0.36
Job 3	0.35	0.07	0.33	0.33	0.31
Weight	20%	10%	30%	40%	

TABLE 2.3: Improved version of the additive weighting method.

In the second attempt, the ranking of the alternatives has changed. This is caused by normalizing the scores before weighting. One problem remains unsolved though: the value scale for interest and opportunity criteria does not capture the distance between two successive values. The used scale is an ordinal scale, i.e. the elements of the scale have a well-defined order but the distance between elements is undefined. Scoring models are not suited to convert qualitative criteria into numbers.

2.3.2 Analytic Hierarchy Process (AHP)

The second method is the Analytic Hierarchy Process (AHP) [33]. This method, developed by Thomas L. Saaty, is widely used in multi-attribute decision making. The method can be used with both qualitative and quantitative criteria and calculates priorities (or weights) from pairwise comparison matrices by using the eigenvalue problem.

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 importance	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 2.4: The fundamental scale for pairwise comparison[33], [34]

The analytic hierarchy process consists of two stages. During 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" [34]. Organizing the factors in such a 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 [32].

During the second stage, both criteria and alternatives are compared in pairs. Every pairwise comparison results in a number that expresses the relation between two criteria or alternatives. Saaty [34] proposes a fundamental scale (listed in Table 2.4) for expressing these relations. The results of the pairwise comparisons are combined in matrices and the analytic hierarchy process then establishes a ranking of the alternatives by calculating the principal right eigenvector of these comparison matrices. The next paragraphs will present an intuitive justification of the method.

Consider n alternatives $A_1, A_2, \ldots, A_n$ and a quantitative criterion X, i.e. X can be expressed with an exact number. The value of a certain alternative A_i with respect to criterion X is given by w_i. For instance, A_i could be n stones and X could be their weight, measured in grams. The ratios between any two alternatives (with respect to property X) can now be summarized in a matrix

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

Note that (1) A is reciprocal, i.e. $a_{ij} = 1/a_{ji}$ and (2) the elements on the diagonal are ones. 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 $\mathbf{w} = (w_1, w_2, \ldots, w_n)^T$, the multiplication would yield a row of identical entries $w_i, w_i, \ldots, w_i$. This is only true in the case of perfect information. In most multi-criteria decision making problems, information is incomplete, incorrect or it simply cannot be represented by numbers (because the criterion describes a qualitative property) and the exact ratios cannot be calculated. In that case, the ratios $a_{ij} = w_i/w_j$ can be estimated by an expert, who is allowed to make (small) errors in judgement. These estimates a'_{ij} make up a new matrix A'. Then, the result of the aforementioned multiplication is a row of statistically scattered values around the actual value w_i. Therefore, it seems reasonable to represent w_i by the average of these values:

$$w'_i = \frac{1}{n} \sum_{j=1}^{n} a'_{ij} w'_j \quad i = 1, 2, \ldots, n$$

This resulting system can be solved for $\mathbf{w}' = (w'_1, w'_2, \ldots, w'_n)^T$ if n is allowed to change. The problem then reads:

$$w'_i = \frac{1}{\lambda_{max}} \sum_{j=1}^{n} a'_{ij} w'_j \quad i = 1, 2, \ldots, n$$

which is equivalent to the eigenvalue problem $A'\mathbf{w}' = \lambda_{max}\mathbf{w}'$. The eigenvector $\mathbf{w}'$, which is unique up to within a multiplicative constant, has to be normalized by dividing its entries by their sum to obtain a *priority vector*, i.e. a vector of appropriate weights or scores for the considered alternatives.

The eigenvalue problem has a number of interesting properties. Reconsider A. This is a positive, reciprocal matrix and it is also consistent, i.e. all entries satisfy the condition

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

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 2.5: Random indices (RI) for reciprocal matrices of order 1 to 9

In other words, the alternatives form a chain which also represents their ranking. Because of this property, all rows of A are linearly dependent and consequently, A has rank one. This also implies that all eigenvalues are zero, except one. In every matrix, the sum of eigenvalues is equal to the sum of elements on the diagonal (also known as the trace of a matrix). Hence, $tr(A) = n$, the order of A.

In the presence of small inconsistencies (where A is approximated by A'), $\mathbf{w}'$ is a good estimate of $\mathbf{w}$ as long as A' is reciprocal because it can be shown that small perturbations of a_{ij} do not lead to large perturbations of λ_{max} and $\mathbf{w}'$ [33]. However, the amount of inconsistency should be measurable in order to decide whether it is small or not. It turns out that the amount of inconsistency is proportional to the difference between the expected eigenvalue n and the actual eigenvalue λ_{max}.

Saaty [33] defines a number, called the *consistency index*, $CI = (\lambda_{max} - n)/(n-1)$, which measures the amount of inconsistency in a given judgement matrix. This consistency index is then compared to the average consistency index of a large number of random reciprocal matrices of the same order, called the *random index* (RI). These random indices can be found in Table 2.5. If the *consistency ratio* $CR = CI/RI$ is 0.1 or less, the estimate $\mathbf{w}'$ is accepted. Otherwise, new judgements should be collected in order to improve consistency.

The strengths of the AHP are that it (1) enables decision makers to structure a problem into a hierarchy, (2) provides a powerful approach for handling both quantitative and qualitative criteria and (3) can deal with inconsistencies. The weaknesses of AHP are that it (1) is a time-consuming method due to the large number of pairwise comparisons and (2) is susceptible to rank-reversal, i.e. the ranking may change when new alternatives are added [29].

EXAMPLE In this example, Ethan uses the AHP method to assist him in choosing a job offer. The goal, which is the root node of the hierarchy, is obviously to select a job offer. The four criteria (salary, distance to work, interest and career opportunities) are children of the root node and the alternatives are children of the criterion nodes. Figure 2.6 illustrates the hierarchy.

First, Ethan has to decide upon the relative importance of every combination of two criteria. The comparison matrix is a 4-by-4 matrix but the diagonal carries only ones and only half of the remaining judgements is needed because the matrix must be reciprocal. Hence, 6 pairwise comparisons have to be made. Ethan uses Saaty's fundamental scale (see Table 2.4) to rate the criteria. For instance, Ethan is convinced that the salary

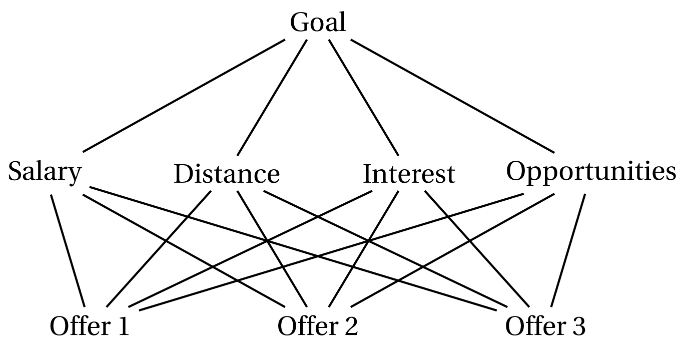

FIGURE 2.6: The decision hierarchy of the running example.

is more important than the distance, which is illustrated in Table 2.6 together with the other scores. The consistency ratio is less than 0.1

	Salary	Distance	Interest	Opportunities	Priority vector
Salary	1	5	$1/3$	$1/3$	0.1525
Distance	$1/5$	1	$1/7$	$1/7$	0.0450
Interest	3	7	1	$1/3$	0.2919
Opportunities	3	7	3	1	0.5106
					$\lambda_{max} = 4.2281$, $CI = 0.0760$, $CR = 0.08$

TABLE 2.6: Comparison of criteria

Second, Ethan has to score the alternatives in a similar fashion with respect to the criteria. The first two criteria are quantitative criteria and these numbers are used to form the ratios. In this case, the comparison matrix is both reciprocal and consistent. Consequently, the consistency ratio is zero. The last two criteria are qualitative criteria and are scored using Saaty's scale. The judgements and resulting priority vectors are listed in Table 2.7.

Finally, all scores are combined into a total score a_i by calculating

$$a_i = \sum_{i=1}^{4} c_i w_i$$

where c_i is the priority of a particular criterion and w_i is the priority of the alternative with respect to this criterion. The vector of total scores is then $(0.3067, 0.4543, 0.2389)^T$.

2.3.3 Fuzzy MCDM

In 1965, Zadeh [35] introduced the concept of fuzzy sets. Five years later, Bellman and Zadeh [36] applied this concept to multi-criteria decision making problems, which allows decision makers to express constraints like "the cost of A should be substantially lower than α".

Salary	J1	J2	J3	Priority
J1	1	$^{2000}/_{2800}$	$^{2000}/_{2600}$	0.2703
J2	$^{2800}/_{2000}$	1	$^{2800}/_{2600}$	0.3784
J3	$^{2600}/_{2000}$	$^{2600}/_{2800}$	1	0.3513
				$\lambda_{max} = 3$, $CI = 0$

Distance	J1	J2	J3	Priority
J1	1	$^{30}/_{5}$	$^{60}/_{5}$	0.8000
J2	$^{5}/_{30}$	1	$^{60}/_{30}$	0.1333
J3	$^{5}/_{60}$	$^{30}/_{60}$	1	0.0667
				$\lambda_{max} = 3$, $CI = 0$

Interest	J1	J2	J3	Priority
J1	1	7	3	0.6694
J2	$^{1}/_{7}$	1	$^{1}/_{3}$	0.0880
J3	$^{1}/_{3}$	3	1	0.2426
$\lambda_{max} = 3.0070$, $CI = 0.0035$, $CR = 0.006$				

Opportunities	J1	J2	J3	Priority
J1	1	$^{1}/_{7}$	$^{1}/_{5}$	0.0668
J2	7	1	5	0.7147
J3	5	$^{1}/_{5}$	1	0.2185
$\lambda_{max} = 3.1827$, $CI = 0.0914$, $CR = 0.16$				

TABLE 2.7: Comparison of job offers with respect to the four criteria

The approach is based on the observation that "most real world decision making processes take place in an environment in which the goals, the constraints and consequences of possible actions are not known precisely" [36]. Usually, this imprecision is associated with randomness, which is fundamentally different from fuzziness. Fuzziness is a type of imprecision that is associated with fuzzy sets. "A fuzzy set is a class of objects in which there is no sharp boundary between those objects that belong to the class and those that do not" [36]. The set of blue objects is an example of a fuzzy set. More formally, if $X = \{x\}$ denotes a set of alternatives, then the fuzzy set A in X

$$A = \{(x, \mu_A(x))\}, \quad x \in X$$

is a set of ordered pairs where $\mu_A(x)$ is called the membership degree of x in A. $\mu_A : X \rightarrow M$ is a function from a space X to a membership space M where $M = [0,1]$ [36].

A decision process is typically comprised of three ingredients: a set of alternatives, a set of constraints and a performance function that indicates the gain (or loss) associated with each alternative. In a fuzzy environment, the goals and/or constraints make up classes of alternatives that do not have crisp boundaries. A fuzzy decision is then an intersection of the given goals and constraints. More formally, if $X = \{x\}$ is a given set of alternatives, then a (fuzzy) goal G is given by a fuzzy set G in X. The membership function $\mu_G(x)$ indicates to which degree a particular alternative x satisfies the goal G. Such a function Similarly, a (fuzzy) constraint C is given by a fuzzy set C in X and the degree to which a certain alternative x satisfies the constraint C is given by $\mu_C(x)$.

The decision is then an intersection of the goal and the constraint:

$$D = G \cap C$$

where D is a fuzzy set, resulting from the intersection of G and C. Correspondingly, the membership function $\mu_D(x) = \mu_G(x) \wedge \mu_C(x)$ is a function of the membership functions of G and C. In case of intersection, $\mu_D(x) = min(\mu_G(x), \mu_C(x))$.

More generally, a multi-criteria decision making problem can have n goals $G_1, \ldots, G_n$ and m constraints $C_1, \ldots, C_m$ and the resulting decision is the intersection of goals and constraints:

$$D = G_1 \cap \ldots G_n \cap C_1 \cap \ldots \cap C_n$$

In fact, other interpretations of "intersection" can be used as well. The decision is then called a "confluence" of goals and constraints.

The strengths of the fuzzy MCDM approach are that (1) linguistic terms can be used for evaluation and (2) improves decision making in the presence of vagueness and ambiguity. The weakness of the approach is that it does not specified how the membership functions should be defined.

EXAMPLE The running example has one goal and four criteria. These can be translated into four goals (great salary, nice driving distance, great job content, great career opportunities) and four constraints (affordable salary, satisfiable distance to work, satisfiable job content, satisfiable job opportunities). That is, every criterion defines both a goal and a constraint. Next, the membership functions should be defined. However, it is not specified how to define these functions. For the sake of simplicity, this example this example will only discuss two membership functions of one criterion. Assume that Ethan thinks that the tipping point for an affordable salary is 1900 euro. The membership function could then be defined as:

$$\mu_{\text{Affordable Salary}}(x) = \frac{1}{1 + 1.05^{1900-x}}$$

Similarly, the tipping point for a great salary is 2800 euro but is also spread out over a larger range:

$$\mu_{\text{Great Salary}}(x) = \frac{1}{1 + 1.005^{2800-x}}$$

Both functions are visualized in Figure 2.7. The membership degree of the decision can now be calculated as the minimum of these two functions:

$$\mu_D(2000) = 0.00035$$
$$\mu_D(2800) = 0.5$$
$$\mu_D(2600) = 0.12025$$

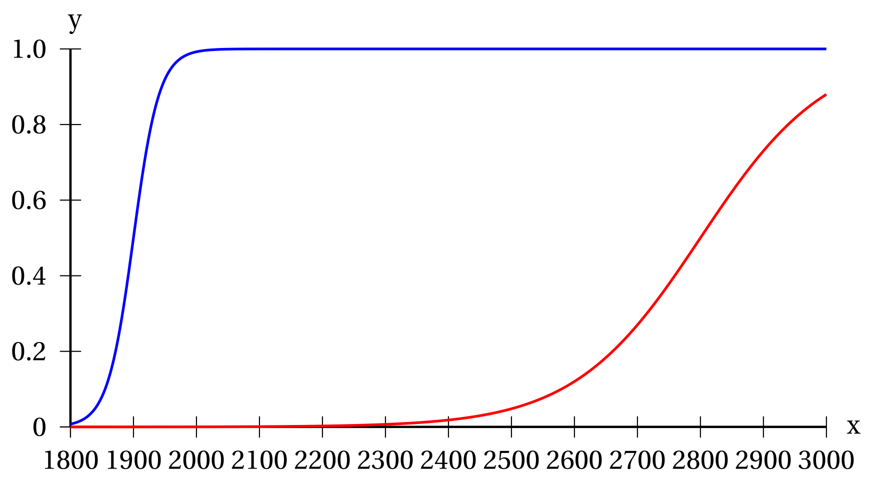

FIGURE 2.7: Visualization of $\mu_{\text{Affordable Salary}}$ (blue) and $\mu_{\text{Great Salary}}$ (red)

Chapter 3

Methodology

This chapter presents the methodology that is used to evaluate and select cross-platform tools. The methodology is inspired by the generic software selection methodology of Jadhav and Sonar [30] (see Section 2.2). This methodology, which comprises of six generic stages.

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

The following section will elaborate on each of these stages.

3.1 Define selection criteria

In the first stage, the selection criteria are defined. Jadhav and Sonar [30] define these as the functional and non-functional requirements that a software package has to meet and they will be used in the third stage 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 obtained during the initial meeting. In order to qualify as a viable cross-platform tool, it has to meet the following requirements:

- **The tool *must* produce "native" Android *and* iOS applications.** CapGemini focusses mainly on Android and iOS because their clients want to support these platforms. Support for other platforms is desirable but not necessary.

- **The tool *must* be able to produce *both* tablet and smartphone applications, preferably from the same codebase.** Depending on the type of application, clients want tablet applications, smartphone applications, or both.

- **The tool *should* be capable of creating enterprise applications.** CapGemini specializes in the development of data-driven enterprise applications. Such applications usually contain a lot of forms and do not require high performance graphics in contrast to 3D games. Even though it is possible to develop an enterprise application on top of a powerful 3D engine, doing so will generally result in lower productivity.

- **The tool *should* have a certain degree of maturity.** If an application is going to be developed (and maintained) with the tool, it should not constantly change its API or have an uncertain future.

- **The *should* provide good support, either by its vendor or by the community.** It is unacceptable for a software project to be delayed because of unresolvable issues. For this kind of problem, the company that is developing the software would like to fall back on an active community or on commercial support offered by the vendor.

3.2 Identify potential candidates

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

Discovery of cross-platform tools has already been done extensively by VisionMobile[1] [25] and research2guidance[2]. VisionMobile has investigated the use of cross-platform tools with more than 2400 developers across 91 countries. Their report contains a list of 100 cross-platform tools that they tracked as part of their research. The research of research2guidance is still ongoing but their questionnaire includes a list of 114 cross-platform tools. Because subsequent Internet searches did not reveal any other tools, the combination of both lists is used as output for this stage. The complete list is available as an appendix (Appendix C).

3.3 List selected alternatives

The candidates obtained from the previous stage are filtered with the selection criteria obtained in 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, including tools that do not meet the selection criteria. Using these, a large number of tools can be omitted.

[1]VisionMobile is an ecosystems analyst firm in the telecom industry, `http://www.visionmobile.com`

[2]research2guidance is a German research and consultancy firm, specialized in the mobile industry `http://www.research2guidance.com`

- Tools that do not produce native applications for Android and iOS, e.g. jQuery Mobile, Sencha Touch, Kendo UI, etc.
- Tools that do not produce tablet and smartphone applications, e.g. Magmito, CrossMob, etc.
- Special-purpose tools that are not well suited for the intended use, e.g. specialized 3D engines like Corona, Marmelade, Unity, Unreal, etc.
- Tools with an uncertain future, e.g. Flash-based systems or cutting-edge tools;
- Tools that lack an active community or do not offer commercial support, e.g. ApPEasy, AppsGeyser, etc.

From the remaining tools, 7 tools (see Table 3.1) are prioritized because of their popularity and focus (enterprise applications).

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 prioritized tools after applying the selection criteria.

This thesis will evaluate two of these tools, together with the native development kits for both Android and iOS. In agreement with Capgemini, two tools are selected: Apache Cordova (also known as PhoneGap) and Motorola Rhodes, which are described in detail in Chapter 4. Apache Cordova is selected because of its popularity and Motorola Rhodes has a strong focus on enterprise applications. Kony also focusses on enterprise applications but has no free trial. Trigger.io is not selected because it is too similar to Apache Cordova, which is the preferred one. MoSync is least prioritized and therefore not selected. The remaining two, Appcelerator Titanium and Xamarin are evaluated by another student with the same research topic.

3.4 Define evaluation criteria

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

3.4.1 Interviews

A finished application is the result of a cooperation between many people with various roles. Every person experiences mobile application development in a different way. In order to gain a better understanding of these experiences, three interviews are conducted with CapGemini employees. The goal of these interviews is to reveal important factors that should be considered during the evaluation of the cross-platform tools. A developer does the actual programming and can illustrate the current state of practice in (mobile) application development. A mobile architect is involved in the complete lifecycle of an application and can highlight some typical issues that are encountered frequently. A salesman negotiates the contracts with clients and can represent the average client. The following sections summarize the outcomes of the conducted interviews.

Developer

Developers are in charge of the actual programming of applications. They can provide valuable insight into the development environment, the current state or practice and other 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 using it. The default development environment is built on top of a Windows operating system and the Eclipse[3] IDE. This IDE is preferred because it is fast, efficient, highly customizable, and most of the documentation assumes an Eclipse environment.

CURRENT STATE OF PRACTICE 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. Every night, all unit tests and a number of statical code analysis tools are run on a so-called continuous integration server. A detailed report of the test run is available in the morning.

ATTITUDE TOWARDS NEW LANGUAGES AND TOOLS Learning new tools and languages is not considered problematic as long as it is worth investing in (because it can increase productivity or is requested by clients). Typically, one or more developers familiarize themselves with the tool/language and then teach the other developers during a training session. In case of technical problems (bugs, architectural issues), Capgemini can — most of the time — fall back on its good partnerships to solve the issue.

REQUIREMENTS FOR CROSS-PLATFORM TOOLS For a developer, a cross-platform tool should (1) work properly, (2) not be too complex, (3) have at least decent performance (preferred over ease of use), and (4) be graphically customizable because most clients have specific styling requirements.

[3] http://eclipse.org

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 considered.

FOCAL POINTS IN MOBILE ARCHITECTURES Mobile architectures consist mainly of two major components: a front-end (the mobile application itself) and a back-end (servers and services providing the data for the application). At the back-end side, the most important factor is the communication with other services that 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.

At the front-end side, the most important factors are the degree of integration with the device and the user interface. Users are accustomed to the native look & feel and wish to see new applications with similar styling. For mobile web applications, stylesheet that mimics the native look & feel of the device are preferred. The requirement for native look & feel makes it hard to write applications that have a single codebase, even though this is desirable.

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

- **Adoption** The number of users is often a good indication of the quality of the tool. Popular tools generally have an active community where users help each other with solving technical difficulties.
- **Maturity** A tool should have some maturity and it should have a bright future. It would not be very wise to start a project that relies on a tool that constantly changes or has an undetermined future.
- **Documentation** Good documentation is priceless as it can save numerous hours of digging through code.
- **Support** Customers want their applications to be delivered in time and they are willing to pay extra for vendor support as a means of insurance. In case of technical issues, Capgemini can contact the vendor to solve the problem.
- **Customizability** Clients have specific requirements with regard to user interfaces and the tool should make them feasible. Plugin architectures can also contribute substantially to the customizability of the tool.
- **Supported devices** The tool should support a sufficient number of devices. In a B2B[4] context, devices are often controlled by the organization and the tool will not

[4]Business-to-business, or commerce between businesses

need to support many devices. In a B2C[5] context, the devices are not managed by the organization and consequently, the tool needs to support a lot more devices.

- **Performance** It is believed that there is a trade-off between productivity, portability and performance. This is acceptable as long as the productivity and portability gains justify the performance loss.
- **Tools** Most developers are more productive with the tools they are familiar with.

Salesman

The salesman is responsible for gathering the client's requirements and sells the application when it is finished. Therefore, the salesman is capable of representing the customers and describing their typical 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 marketing agencies that make this kind of applications with a rather short life span.

Capgemini focusses on the second type of applications. These applications mostly handle data and do not rely on device specific hardware like the GPS or accelerometer. 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[6] technology, which rules out iOS as a platform. When there is no compelling reason to choose one or the other, Capgemini can help the client to make an appropriate decision. Performance is almost never a primary requirement.

3.4.2 Literature review

The criteria extracted from the interviews are complemented with criteria from literature. As mentioned 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 [25] and Gartner [37].

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

VisionMobile has conducted a worldwide survey about cross-platform tools. The survey includes the key reasons for both selecting and dropping a particular cross-platform tool. The top 10 reasons for both are summarized in Table 3.2 and Table 3.3.

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

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

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

TABLE 3.3: Top 10 reasons for dropping 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].

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) access to hardware 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.

From this list, (3), (4) and (9) are omitted because (3) and (4) are related to costs and (9) does not apply for Capgemini. Skill reuse (2) and learning curve (5) are merged because they are related to each other: if skills can be reused, the learning curve will be reduced and vice versa.

Gartner has also done similar research in 2011 [37]. The report adds (13) toolset reuse and (14) code reuse as additional criteria and the criteria are organized in three categories: portability, native experience and development experience.

3.4.3 Evaluation criteria

The evaluation criteria, identified from the interviews and literature review, are combined into a single hierarchy, which comprises of three top-level categories.

- **Portability** describes the ability to reuse certain assets of a project when migrating to another platform.
  - **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. Both literature [25], [37] and interviews mention this criterion.
  - **Toolset reuse** describes the reuse of development tools like IDE's, operating systems, etc. This criterion is extracted from Gartner [37].
  - **Code reuse** expresses the amount of code that can be reused across platforms. This criterion is also extracted from Gartner [37].
- **Application Experience** describes the experience of the finished product: integration with the device, user interface and performance.
  - **Native Integration** describes the integration with the platform.
    - **Access to hardware** describes the quality and extensiveness of device APIs. This criterion can be found in both literature and interviews.
    - **Platform-specific services** describes integration with platform-specific services like push notifications. Cross-platform tools could follow a "least common denominator" approach in which only common functionality is offered and flexibility is sacrificed. This criterion is coming from Gartner's report [37].
  - **UI capabilities** describes the ability to create a rich user interface.
    - **Native Look & Feel** describes whether native user interface elements can be used. This criterion is mentioned in the interviews and in the Gartner report [37].
    - **UI element capabilities** describes the capabilities of user interface elements. This criterion originates from literature [25], [37] and interviews.

  - **Performance** describes the overall performance of the application, like the snappiness of the user interface. It is referenced in both literature [25], [37] and interviews.

- **Productivity** is a measure for the ease of development.
  - **Skill reuse** describes the ability to reuse existing skills, which will reduce the learning curve. This criterion is found in literature [25], [37] and interviews.
  - **Tooling** focusses on the quality of the development environment and is mentioned in literature [37] and interviews.
  - **Testing** measures the quality of software testing tools and originates from the interview with the developer.

3.5 Evaluate selected alternatives

In this stage, the actual evaluation of the selected tools takes place. The quality of the tools is estimated from hands-on experience with each of the discussed tools. The tools are used to develop a simple proof-of-concept application which is described in Section 3.5.1. How the experience is then used to evaluate the tools is further described in Section 3.5.2. The actual evaluation is presented in Chapter 5.

3.5.1 Proof-of-Concept application

The proof-of-concept application is, a rather small, typical enterprise application that contains the most requested features. This helps to ensure that the selected tools are thoroughly tested with regard to these essential features.

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. A mobile app can significantly reduce the processing time of these refunds.

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 enterprise application elements

The proof-of-concept application includes a number of features that are typically required in any business application. The requirements documentation is available in Appendix D.

- **User interface elements** 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** Tablets can make better use of their screen real estate. Therefore, their layout often consists of a narrow column on the left and a wide column on the right (this is also known as the master-detail interface).
  - **Phone UI** Smartphones have smaller screens. Hence, the master and detail views are decoupled in separate views and connected through navigation.
- **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 does not have an active internet connection.

- **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.

Note that in most cases, specific non-existing functionality can be implemented on top of other existing functionality. However, this requires extra work and a cross-platform tool is more interesting if it provides all the functionality readily out of the box.

3.5.2 Evaluation methodology

For the actual evaluation, the analytic hierarchy process (AHP) [33] (see Section 2.3.2) is used. This method assigns weights to criteria and alternatives based on eigenvector calculations. This method is selected for multiple reasons:

- The evaluation is entirely based on relative, pairwise comparison instead which is something that humans are good at.
- It can deal with both quantitative and qualitative criteria. Most of the evaluation criteria are qualitative criteria which would have to be quantified when using scoring models. Unlike AHP, scoring models do not provide a mechanism for detecting inconsistencies when doing so.
- It is unclear from literature how membership functions are defined in fuzzy MCDM.
- AHP has been successfully applied multiple times [29], [30]

The evaluation has two key ingredients: the evaluation criteria and the cross-platform tools or alternatives. The total score of each alternative is calculated in three steps. First, a weight is assigned to every criterion. Second, a weight is also assigned to every alternatives with respect to a particular criterion. In order to do so, the criteria (and alternatives) are compared in pairs. The results of these comparisons are combined in a matrix from which the weights can be calculated using. Last, these weights are combined to calculate the total score of each alternative. If there are n alternatives and m criteria, the total score 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 a criterion C_i and w_i is the weight an alternative A_j with respect to criterion C_i. The total scores can then be used to rank the alternatives. The actual evaluation of the cross-platform tools is presented in Chapter 5.

3.6 Select the most suitable alternative

The final step of the software selection process is the selection itself. The final scores of the alternatives, obtained in the evaluation stage make up a ranking. However, having the best total score does not imply that an alternative is the most cost-effective. In the fourth stage, criteria related to costs were deliberately left out. Separating costs and benefits enables the decision maker to perform a cost-benefit analysis. This analysis might reveal that other alternatives are more cost-effective while still having acceptable benefits.

Chapter

4 About the evaluated tools

In this chapter, the evaluated tools are described in more detail together with information about how they are used to implement the proof-of-concept application.

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 the 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. 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.

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

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

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

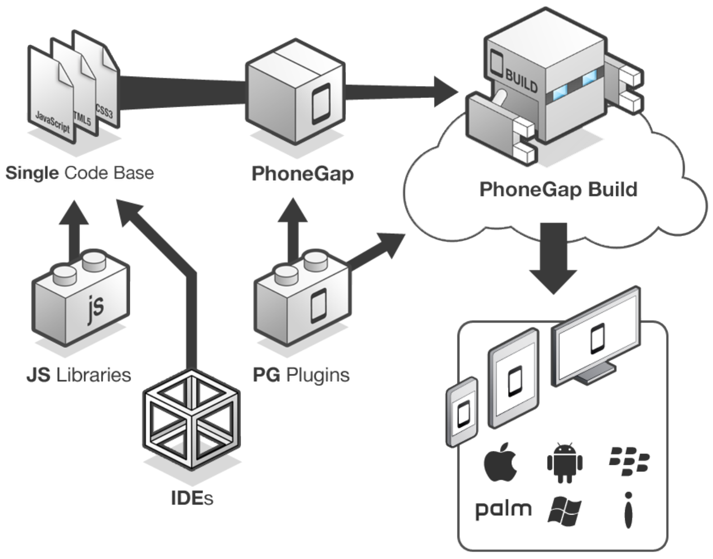

FIGURE 4.1: The architecture of Apache Cordova/Adobe PhoneGap applications [38].

4.1.2 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. The architecture is presented 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 sufficient HTML5 support. Cordova solves this problem by providing HTML5 polyfills for these browsers. A polyfill is a replacement implementation for missing functionality. The API of the polyfill typically matches the official HTML5 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. For 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

[3] http://dev.w3.org/html5/webdatabase/
[4] http://dev.w3.org/html5/webstorage/

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 online[5]. However, at the time of writing, there is no package manager available for Cordova plugins, like NPM[6].

4.1.3 Related projects

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

WEINRE

WEINRE[7] 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[8] server. The only requirement is to include a javascript file in the application's HTML file. 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, and run custom JavaScript commands remotely. A hosted version is available online[9].

Apache Ripple

Apache Ripple[10] 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.

Adobe PhoneGap Build

Adobe PhoneGap Build[11], 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[12] 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.

[5] `https://github.com/phonegap/phonegap-plugins`
[6] NPM is the Node Package Manager, `http://npmjs.org`
[7] `http://people.apache.org/~pmuellr/weinre/docs/latest/`
[8] Server-side JavaScript, `http://nodejs.org`
[9] `http://debug.phonegap.com`
[10] `http://ripple.incubator.apache.org`
[11] `http://build.phonegap.com`
[12] Git is a popular version control system, `http://git-scm.org`

Edge Inspect

Another Adobe product is called Edge Inspect[13], 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 actually based on WEINRE). The application also allows the developer to take screen shots the web pages displayed in a particular browser.

4.1.4 Implementation of 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. The application logic is built with AngularJS, the user interface is built on top of Bootstrap. 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. The exact role of each tool is discussed in the following sections.

AngularJS

AngularJS[14] 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 testing easier. For instance, components that require user interaction or are not useful in a testing environment. Using dependency injection, they can be replaced easily with a mock. A mock is a replacement component that mimics the behaviour of the element it replaces but it does not rely on components that are not available in a testing environment.

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 18 man months and 17000 lines of traditional HTML and JavaScript code, could be completely rewritten using AngularJS in three weeks by a single individual and with only 1500 lines of code [39].

[13] http://html.adobe.com/edge/inspect/
[14] http://angularjs.org

Bootstrap 3

Bootstrap[15] is a popular HTML user interface framework that came to life at Twitter. The latest revision, (version 3, still in beta) 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[16] is a collection of tools and best practices to make web development easier. It is comprised of the following tools:

- **Yo** is a command-line tool used for scaffolding application code. A Yo plugin for AngularJS projects is available online[17] and can generate various application artifacts like a basic project setup, controllers, services, directives, etc.
- **Grunt** is a command-line tool, similar to the make program[18]. It can run various predefined JavaScript tasks: compiling LESS files, minifying javascript and CSS, linting Javascript, running unit tests, etc.
- **Bower** is a package manager for web applications and takes care of tracking complex dependency graphs in a project. It is available as a command line tool.

Jasmine & Karma

One of AngularJS' focal points is writing testable code. The community has created a test runner, called Karma[19]. 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[20]) 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[21] testing framework.

4.2 Motorola Rhodes

4.2.1 General information & history

Rhodes[22] 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

[15] http://getbootstrap.com
[16] http://yeoman.io
[17] https://github.com/yeoman/generator-angular
[18] https://www.gnu.org/software/make/
[19] http://karma-runner.github.io/
[20] http://phantomjs.org/
[21] http://pivotal.github.io/jasmine/
[22] http://docs.rhomobile.com

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.

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).

4.2.2 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.

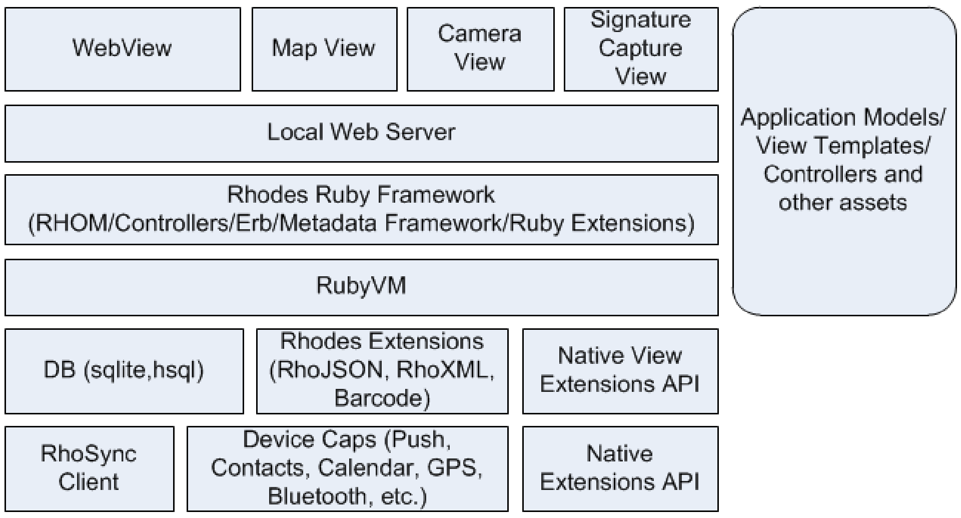
WebView
Map View
Camera View
Signature Capture View
Local Web Server
Rhodes Ruby Framework (RHOM/Controllers/Erb/Metadata Framework/Ruby Extensions)
RubyVM
DB (sqlite,hsql)
Rhodes Extensions (RhoJSON, RhoXML, Barcode)
Native View Extensions API
RhoSync Client
Device Caps (Push, Contacts, Calendar, GPS, Bluetooth, etc.)
Native Extensions API
Application Models/ View Templates/ Controllers and other assets

FIGURE 4.2: The architecture of Rhodes applications [40]

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.

4.2.3 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.

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 an 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[23] 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, BlackBerry, Windows Mobile, and Windows.

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.

4.2.4 Implementation of the proof-of-concept application

The proof-of concept application is implemented as a multi-page application because it is the way Rhodes is intended to be used. The downside of this approach is it introduces waiting time for every page it loads. The user interface is also based on Bootstrap.

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

[23] `https://app.rhohub.com/`

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 the evaluation of the selected alternatives, which are the combination of native Android and iOS, Apache Cordova and Motorola Rhodes. First, the weights of the evaluation criteria are calculated (Section 5.1). Second, the weights of the selected alternatives are calculated (Section 5.2), and last, a cost-benefit analysis is performed (Section 5.4).

5.1 Weighting the evaluation criteria

The weights for the evaluation criteria are calculated from judgements given by a developer and a mobile architect at Capgemini (the same people that gave the interviews in 3.4.1). The judgements are collected from a questionnaire containing all necessary pairwise comparisons (see Appendix **??**). The developer and the architect do no necessarily share the same opinions about the evaluation criteria. Therefore, this evaluation covers these two perspectives separately as they could have a different ranking of the alternatives.

The weights (elements of the priority vector) are calculated as follows. The children of every non-leaf node in the evaluation criteria tree (visualized in Figure 5.1), are compared in pairs. For instance, the root node has three children, resulting in three pairwise comparisons: portability vs. application experience, application experience vs. productivity, and productivity vs. portability. The relative importance of two criteria is scored using Saaty's fundamental scale [33], [34] (see Table 2.4). For instance, if the architect strongly favours productivity over portability, a 5 is written in the comparison matrix for the element productivity vs. portability (see Table 5.1). From this matrix, the principal right eigenvector is calculated and normalized[1]. The consistency index is calculated from the eigenvector. More information about AHP can be found in Section 2.3.2. Finally, the actual weight of every node is calculated as the product of all nodes that make up the path from the root node to that particular node.

The following sections discuss the gathered judgements of the evaluation criteria.

[1]In AHP, a vector is normalized by dividing its elements by their sum.

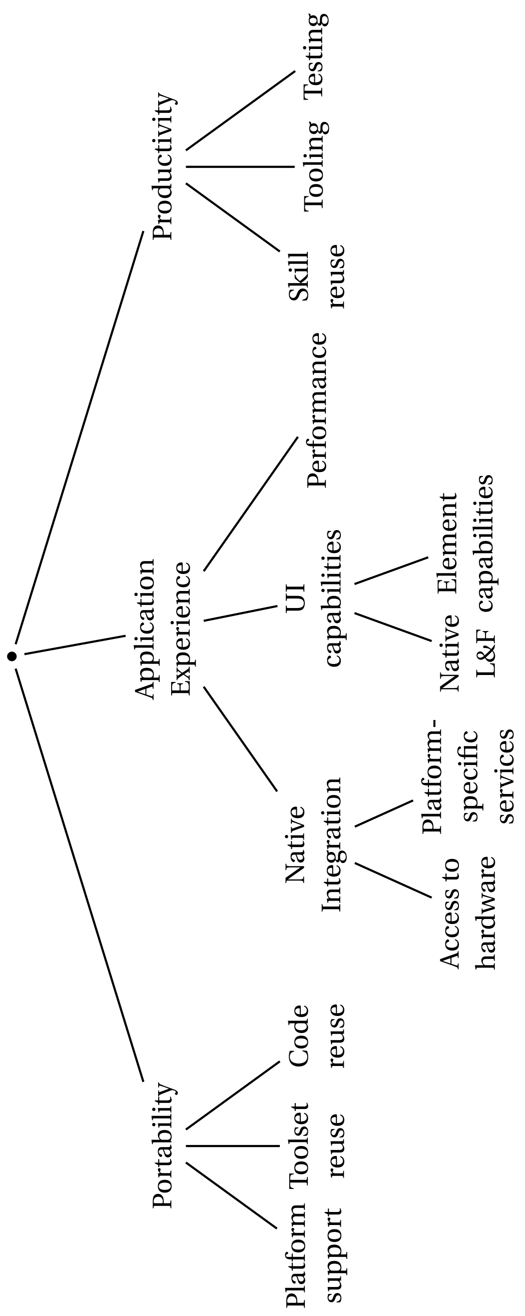

FIGURE 5.1: The evaluation criteria, organized in a tree

5.1.1 Top-level evaluation criteria

The tree is decomposed into three top-level evaluation criteria: “portability”, “application experience”, and “productivity”. The weight distribution for these criteria, presented in Table 5.2, is similar for both the architect and developer. There is, however, a large inconsistency in the judgement of the architect ($CR > 0.1$). In spite of the high inconsistency, the judgements are accepted because no new judgements could be gathered in time. Preferably, new judgements should be gathered until the consistency is satisfiable.

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.

5.1.2 Portability

The first top-level evaluation criterion, “portability”, is decomposed into “platform support”, “toolset reuse”, and “code reuse” and the judgements of the developer and architect are illustrated in Table 5.2. Here, the differences between the developer and architect are a bit more pronounced but again, high consistency is observed in the architect’s judgement. However, no better judgements were available.

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”.

5.1.3 Application Experience

The second top-level criterion, “application experience”, is decomposed into “native integration”, “user interface capabilities”, and “performance”. The gathered judgements from both developer and architect are listed in Table 5.3. Here, the differences are also pronounced. While the developer mainly focusses on performance, the architect prefers a mix of native integration and performance. Again, high inconsistency is noted in the architect’s judgement.

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”.

The criterion “native integration” is further broken down into “access to hardware” and “platform-specific services”. Here, the developer and architect have clearly different opinions, as is visible in Table 5.4. Because there are only two subcriteria, there cannot be any inconsistency.

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”.

The criterion “user interface capabilities” is also divided into two subcriteria: “native look & feel” and “user interface element capabilities”. Also here, the developer and architect have opposing opinions which is shown in Table 5.5. The architect strongly prefers native look and feel over element capabilities while it is the other way round for the developer. This has been observed earlier in the interviews as well (see Section 3.4.1). Again, there is no inconsistency because there are only two subcriteria.

5.1.4 Productivity

The last top-level criterion “productivity” is decomposed into “skill reuse”, “tooling”, and “testing”. Apart from small deviations in judgement, the focus is mainly on tooling and skill reuse as can be seen from Table 5.6. Here, the inconsistency in the architect’s judgement is still high but rather acceptable.

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".

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".

5.2 Evaluate the alternatives

In the subsections to follow, all alternatives are evaluated with respect to one criterion at a time. Every subsection gives a description of what is measured and how. 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 that specific property. In that case, judgements based on pairwise comparison, as proposed by Saaty [33], are used in order to evaluate the alternatives. These judgements arise from the experience of developing the proof-of-concept application. In addition, some criteria deal with features that are not addressed in the proof-of-concept application. In that case, the evaluation is based on the information found in the documentation of the different tools.

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

5.2.1 Platform support

As mentioned in Section 3.1, support for Android and iOS is a selection criterion. Hence, this criterion only covers additional platforms, in addition to 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 (the sum of scores is equal to 1). The market share numbers are taken from research by Gartner [1]–[16] (see 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, 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.

Applications that are built with the native SDK's can only be used on their intended platforms. Cordova and Rhodes can reach additional platforms, which is shown in Table 5.7.

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".

5.2.2 Toolset reuse

This criterion describes the ability to reuse the development environment or parts thereof (1) for the development for both platforms and (2) when migrating to another cross-platform alternative. The evaluation is based on the recommended development environment setup for each alternative. Since there is no absolute scale available to measure this property, Saaty's fundamental scale (see Table 2.4) is used here.

ANDROID & IOS All iOS development is done with the Xcode IDE, which is available on Mac OS X only. Every iOS developer must therefore own an Apple computer. For Android development, any operating system can be used in combination with the Android SDK and an Eclipse IDE for which a plugin, called Android Development Toolkit (ADT), is available. On the latest developer conference, Google also announced a new and freely available Android IDE, based on JetBrains' IntelliJ IDEA[2].

APACHE CORDOVA For HTML5 application development, any operating system with a web IDE or even a simple text editor is sufficient. However, packaging the Cordova appli-

[2]IntelliJ IDEA is another Java IDE, comparable to Eclipse, http://jetbrains.com/idea

cation in its native shell requires the native SDK's or an online build service like PhoneGap Build. These online build services enable developers with Windows or Linux operating systems to develop applications for iOS. Unfortunately, this particular service cannot build additional plugins that are not part of the standard distribution. If a project makes use of custom plugins, the environment has the same requirements as that of the native SDK's.

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 an online build service, like RhoHub.

The development environments for Rhodes and Cordova are quite similar and are treated equally in the comparison for that reason. Android and iOS are treated as a whole and because of Apple's strict requirements, it is more limited than Rhodes and Cordova. But, if the development environment is based on OS X, then there are no notable differences. This rationale is summarized 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 match existing HTML5 specifications. Therefore, these applications can be easily ported to other, non-mobile platforms. However, non-mobile platforms are out of scope for this evaluation.

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.

The native code for Android and iOS cannot be reused across platforms whereas all the code is reused in Cordova and Rhodes application. This is shown 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 with the device's hardware. The availability of device API's (sensor API's, output API's, and communication API's) of every alternative is compared side by side. An alternative can either provide full support, partial support or no support at all. If an additional plugin system is present, the alternative could provide the same levels of support through a plugin. 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 full access to the device but that does not imply that all these device features are available on a particular device. Apple's iDevices have less device features but provide full support for those that are present.

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" [41]. 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 part of the specification. However, Cordova comes with an elaborate plugin system which can be used to reach the same level of integration as native Android and iOS.

MOTOROLA RHODES Rhodes has a unified sensors API on which one can listen for events coming from all types of sensors if they are present. Rhodes lacks a VideoCapture feature on all platforms and a MediaPlayer on iOS out of the box but these can be implemented using the plugin system.

From Table 5.10, it is clear that all device features can be integrated with, independent of the alternative used. If the alternative does not support a current feature out of the

API	Android	iOS	Cordova	Rhodes
Accelerometer	✓	✓	✓	✓
Geolocation	✓	✓	✓	✓
Gyroscope	✓	✓	full w/ plugin	✓
Light	✓	✓	full w/ plugin	✓
Magnetic Field	✓	✓	✓	✓
Pressure	✓	–	partial w/ plugin	partial
Proximity	✓	✓	full w/ plugin	✓
Relative Humidity	✓	–	partial w/ plugin	partial
Temperature	✓	–	partial w/ plugin	partial
Microphone	✓	✓	✓	✓
Camera	✓	✓	✓	partial
Hardware buttons	✓	–	partial	partial
Vibration Motor	✓	✓	✓	partial
Speaker	✓	✓	✓	partial
System information	✓	✓	✓	✓
File System	✓	✓	✓	✓
Contacts	✓	✓	✓	✓
Bluetooth	✓	✓	full w/ plugin	✓
NFC	✓	–	partial w/ plugin	partial w/ plugin
Wi-Fi Direct	✓	–	partial w/ plugin	partial w/ plugin
USB / Accessory	✓	✓	full w/ plugin	full w/ plugin
Full support	Combined: 15/21		10/21	11/21
Partial support	Combined: 6/21		1/21	7/21
Full support w/ plugin	Combined: 0/21		5/21	1/21
Partial support w/ plugin	Combined: 0/21		5/21	2/21
Total	Combined: 21/21		21/21	21/21

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

box, it can be reached trough a plugin. This requires some additional native coding but the plugin can be reused. Therefore, Cordova and Rhodes are only mildly penalized in Table 5.11.

Access to hardware	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 "access to hardware"

5.2.5 Integration with platform-specific services

This criterion measures the ability to integrate 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 easily implemented in 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.

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 5.12.

Platform-specific services	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 5.12: 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. 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 an 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.

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 score better compared to Cordova and Rhodes, as is illustrated in Table 5.13.

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

TABLE 5.13: 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. In the proof-of-concept application, an additional UI framework, called Bootstrap, was used to create a rich user interface. In this evaluation, alternatives for the typical native user interface elements are searched in the HTML5 specification and the Bootstrap framework.

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[3]. 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[4] 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.

HTML only offers basic user interface building blocks. However, with these blocks, one can create a very rich user interface, e.g. by using `a`'s, `div`'s, list elements and styling one can create create powerful navigation bars. The use of an additional user interface framework like Bootstrap can be a great help in this area. In the end, all alternatives have a more or less equally rich user interface, which is indicated in Table 5.14.

[3] http://angularjs.org
[4] 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 5.14: 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).
- **iPad** A tablet, released in 2010 and running iOS 5.1.1 on a single core 1 GHz ARMv7 processor. It has 256 MB of RAM and a 9.7-inch display with a 768 by 1024 pixel resolution (132 ppi).

ANDROID & IOS The native SDK's are best placed 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 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).

The judgements for this evaluation are listed in Table 5.15. 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 experience with Java and web development (because these are the areas of expertise at Capgemini).

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 5.15: Evaluation of the alternatives with respect to "Performance".

ANDROID & IOS For organizations that already have expertise with Java development, 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.

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 5.16.

Skill reuse	Android/iOS	Cordova	Rhodes	Priority vector
Android/iOS	1	1/5	1	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 5.16: 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 (see Table **??**).

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 ships with command-line interface for building the application. Sadly, this interface poorly integrates with the Android Debug Bridge (ADB) and there is no option to transfer iOS applications to an iDevice. Instead, either Xcode, iTunes or Fruitstrap[5] is needed for the transfer. Next to the command-line tools, Rhodes also ships with an Eclipse-based IDE, called RhoStudio. However, this IDE could not be configured to make it work with the Rhodes SDK.

The experience with the tools during the development of the proof-of-concept application are listed in Table 5.17. The experience with Rhodes was awful and the experience with Cordova was a little less pleasant than with the tools for native Android and iOS.

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 5.17: Evaluation of the alternatives with respect to "Tooling".

5.2.11 Testing

This criterion measures both 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. End-user testing 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, since this is an important topic in a production environment, the basics are examined.

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[6], even headless[7]. For iOS, there are a number of guides available de-

[5]Fruitstrap is a command-line script for transferring applications to an iDevice, https://github.com/ghughes/fruitstrap

[6]Jenkins is a popular continuous integration server, http://jenkins-ci.org

[7]On a server without a user interface

FIGURE 5.2: A radar graph, visualizing the weights of all alternatives

scribing how to set it up. However, Apple hardware is still required. Other options include a hosted continuous integration service.

APACHE CORDOVA For JavaScript, the Jasmine testing framework can be used to write tests (1). In combination with a test runner like Karma, these test can be automated 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.

The judgements are summarized in Table 5.18. All alternatives have support for unit testing. In that perspective, they are all equal but 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 5.18: Evaluation of the alternatives with respect to "Testing".

5.3 Global verdict

The weights of all alternatives with respect to the evaluation criteria is illustrated in Figure 5.2. Now that the weights of both the criteria and alternatives are given, 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 5.19 and Table 5.20 respectively.

The ranking of the alternatives is the same for both. Consequently, 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%.

When comparing the alternatives in pairs and using Saaty's scale (see Table 2.4), one can conclude that (1) Android and iOS are a little better than Cordova (60.8/27.04 = 2.25), (2)

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 5.19: 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 5.20: Summary of the total weight and weighted scores of the alternatives with respect to the developer's weights.

Cordova is a little better than Rhodes (27.04/12.15 = 2.22), and (3) Android and iOS are a clearly better than Rhodes (60.8/12.15 = 5).

5.4 Costs versus benefits

Having the most benefits does not imply that a tool is the most cost-effective as no company will 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 perform a cost-benefit analysis. In this section, the benefits of each alternative will be compared to their costs.

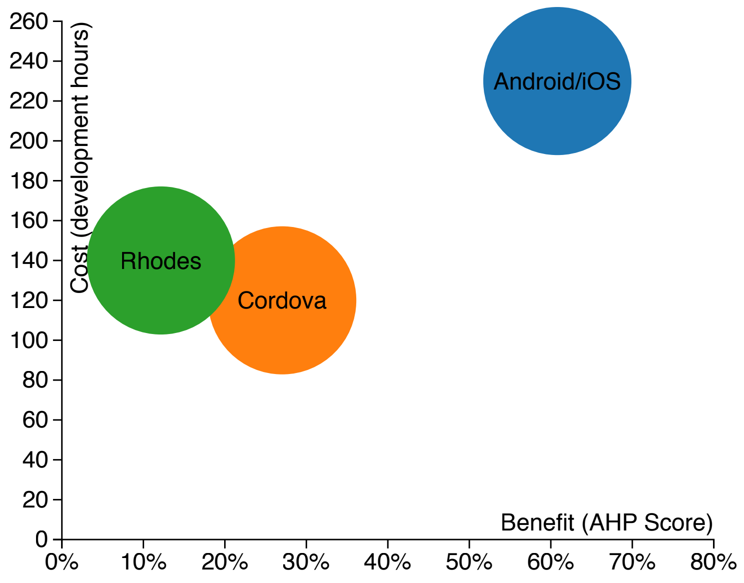

FIGURE 5.3: Visual representation of the cost-benefit analysis. The benefits are indicated on the horizontal axis. The cost of the project is represented by the development time on the vertical axis.

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 5.3.

It is important to mention 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.

Chapter 6

Conclusion

6.1 Goals

This thesis had two major goals. The first goal was to design a methodology to evaluate and select 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 best candidate. The methodology used was 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 presented 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.

Next, a list of potential candidates was composed from Internet searches and literature. An extensive list of cross-platform tools was was composed from “Cross-platform developer tools 2012”, a report by VisionMobile on this subject [25] and ongoing research by research2guidance. 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.

Subsequently, the evaluation criteria were defined from literature review and interviews with a developer, a mobile architect, and a salesman at Capgemini. 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 into a hierarchy with because literature suggests that humans can only deal with 7 plus or minus 2 pieces of information simultaneously [32].

During the fifth stage, the alternatives were evaluated using these evaluation criteria. For this evaluation, the analytic hierarchy process (AHP) [33] was used. This method assigns weights to both criteria and alternatives based on judgements that originate from pair-wise comparisons. In order to formulate a more reliable judgement, a proof-of-concept application was implemented with the studied tools. The relative importance of the evaluation criteria is judged by a developer and a mobile architect. Because a different ranking of the evaluation criteria could lead to a different ranking of the cross-platform tools, the evaluation covers these two perspectives separately.

During the final stage, the candidates that 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, provided that the lost benefits are acceptable.

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. Continuous evaluation is recommended.

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 (mobile) 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. However, having multiple individuals evaluate the alternatives was simply not an option and only two employees were available for questioning. The evaluation is based on the judgement of the author. Hence, the result does not represent the animo among developers and architects but result rather represents a single case of 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 statistically valid results.

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.

The proof-of-concept application was used to gain the experience to formulate a more reliable judgement about the tools. The application itself has not been completed, let alone production ready. The development of the Rhodes application has even been cancelled due to the bad quality of that cross-platform tool. The Cordova application currently wraps all the necessary source code. Instead, future versions could fetch the code from a URL and cache this code using AppCache. This way, the application can be kept up-to-date without the need for updating the outer shell.

Appendices

Appendix

A Scientific Paper

Appendix

B Poster

Appendix

C List of identified cross-platform tools

The list is composed from (A) a completed survey by VisionMobile [25], and (B) an ongoing survey by research2guidance.

cross-platform tool	in A	in B
++Technologies (XPower++)	✓	
5App		✓
Adobe Air	✓	✓
Adobe Flex	✓	✓
alchemo (Innaworks)	✓	✓
Ansca Mobile (Corona SDK)	✓	✓
Antenna Software (Mobility Studio)	✓	
Antix Labs (Games Development Kit)	✓	
Any Presense		✓
App Lifecycle Platform	✓	✓
Appcelerator (Titanium)	✓	✓
AppConKit		✓
AppEasy		✓
Appflight		✓
AppFurnace		✓
Appletta		✓
Application Craft	✓	✓
AppMachine		✓
AppMakr		✓
appMobi (appMobiXDK)	✓	✓
AppQuartz		✓
Apps Geyser		✓
Apps-Builder	✓	✓
Appsbar		✓
Appscend		✓
AppShed		✓
AppStudio		✓
Artech (GeneXus)	✓	✓

cross-platform tool	in A	in B
Backelite (BKrender)	✓	
Battery Powered Games (BatteryTech)	✓	
Bizness Apps		
Brightcove (App Cloud)	✓	✓
Canappi	✓	✓
Capriza		✓
Cocos2D	✓	
CodenameOne		✓
Conduit Ltd (Conduit Mobile)	✓	✓
Construct2		✓
CoStore (PixelSpark)	✓	
CrossMob		✓
DaVinci Studio		✓
Delphi XE3		✓
Department of Behaviour and Logic (Cabana)	✓	
DHTMLX Touch	✓	✓
Didmo (Magmito)	✓	✓
Digital Fruit (Lime JS)	✓	
Dojo Foundation	✓	✓
DragonRAD (Seregon)	✓	✓
EdHouse (IPFaces)	✓	
Elements Interactive Mobile (Edgelib)	✓	✓
ELIPS Studio		✓
Emo	✓	
Enyo		✓
Expanz (Expanz Platform)	✓	
Facebook (Strobe, Sproutcore)	✓	
FeedHenry	✓	
Firemonkey		✓
FlexyCore (In-the-box)	✓	
foneFrame		✓
Game Salad Inc (Game Salad)	✓	✓
Gamebuilder Inc. (Gamebuilder Studio)	✓	
Genero		✓
Geniem (Appever)	✓	
Gideros Mobile	✓	✓
GoCanvas		✓
GTK		✓
Haxe NME	✓	✓
IBM (Worklight)	✓	✓
iBuildApp Inc (iBuild App)	✓	✓
ICEMobile		
Ideaworks 3D Ltd (Marmelade)	✓	✓
iGenApps		✓

cross-platform tool	in A	in B
Illumination Software Creator		✓
ITR Mobility (iFactr)	✓	
iUI	✓	
J2ME Polish	✓	✓
JMango	✓	
Jo	✓	✓
Job and Esther Technologies Ltd (Eqela)	✓	
Joshfire		✓
jQuery Mobile	✓	✓
Kendo UI		✓
Kiahu		✓
Kony (KonyOne Platform)	✓	✓
Kyros (Velocity)	✓	
Lifecycle Mobile (FiveSpark)	✓	
Liquid State		✓
Ludei		✓
Me and my App		✓
Metrosmith		✓
mFoundry		✓
Minimob		✓
MobAppCreator		✓
MobBase		✓
MobiCart		✓
Mobile Nation (Mobile Nation HQ)	✓	✓
Mobile Roadie		✓
MobileDataForce		✓
Mobinex Smartface	✓	✓
MobiOne		✓
Mono		✓
Monocross		✓
MoSync	✓	✓
Motorola Solutions (RhoMobile)	✓	✓
NeoMades (NeoMAD)	✓	
Netbiscuits	✓	✓
Octomobi	✓	
OpenText (Mobile Wave Platform)	✓	✓
Oracle (ADF)	✓	
Page2Flip		✓
Pajap		✓
Pancoda (The M Project)	✓	
Papaya Mobile (Social Game Engine)	✓	
PhobosLab (impact.js)	✓	
PhoneGap (Apache Cordova)	✓	✓
Qt	✓	✓

cross-platform tool	in A	in B
Qualcomm (BREW)	✓	✓
QuickConnect Family	✓	
Raddical Breeze (Illuminations)	✓	
Red Foundry	✓	✓
RunRev (LiveCode)	✓	✓
Sencha (Touch, jQTouch)	✓	✓
SevenVal (FITML)	✓	
ShoutEm		✓
Sideshow NetQuest (Proto.io)	✓	
SIO2 Interactive (SiO2 Engine)	✓	
SmartApp		✓
Software AG (Bedrock)	✓	✓
Spot Specific	✓	✓
SpringSource, VMWare (Grails, SpringMVC)	✓	
StackMob	✓	
Stencyl		✓
Stonetrip (ShiVa 3D)	✓	
SuperWaba (TotalCross)	✓	✓
SwebApps		✓
Sybase (UnWired Platform)	✓	
TapLynx		✓
The Game Creators Ltd (App Game Kit)	✓	
Tiggzi		✓
Trigger Corp (Trigger.io)	✓	
Unity Technologies (Unity)	✓	✓
Unreal (Unreal Engine)	✓	
UX Plus Inc. (Aqua Platform)	✓	✓
V-Play		✓
Vaadin	✓	
Verivo	✓	✓
Vexed Digital (Kirin)	✓	
ViziApps		✓
Webmethods		✓
WebMobi		✓
WeeverApps		✓
Whoop		✓
WidgetBox Mobile		✓
WidgetPat		✓
WM App Builder		✓
wxWidgets	✓	
Xamarin (Xamarin.iOS, Xamarin.Android)	✓	✓
XMLVM	✓	
XUI.js	✓	
YoYO Games (YoYo Games Maker)	✓	

cross-platform tool	in A	in B
ZipLine Games (Moai)	✓	

Appendix

D Requirements documentation of the proof-of-concept application

D.1 Class Diagram

D.2 Mock-ups

D.2.1 Login screen

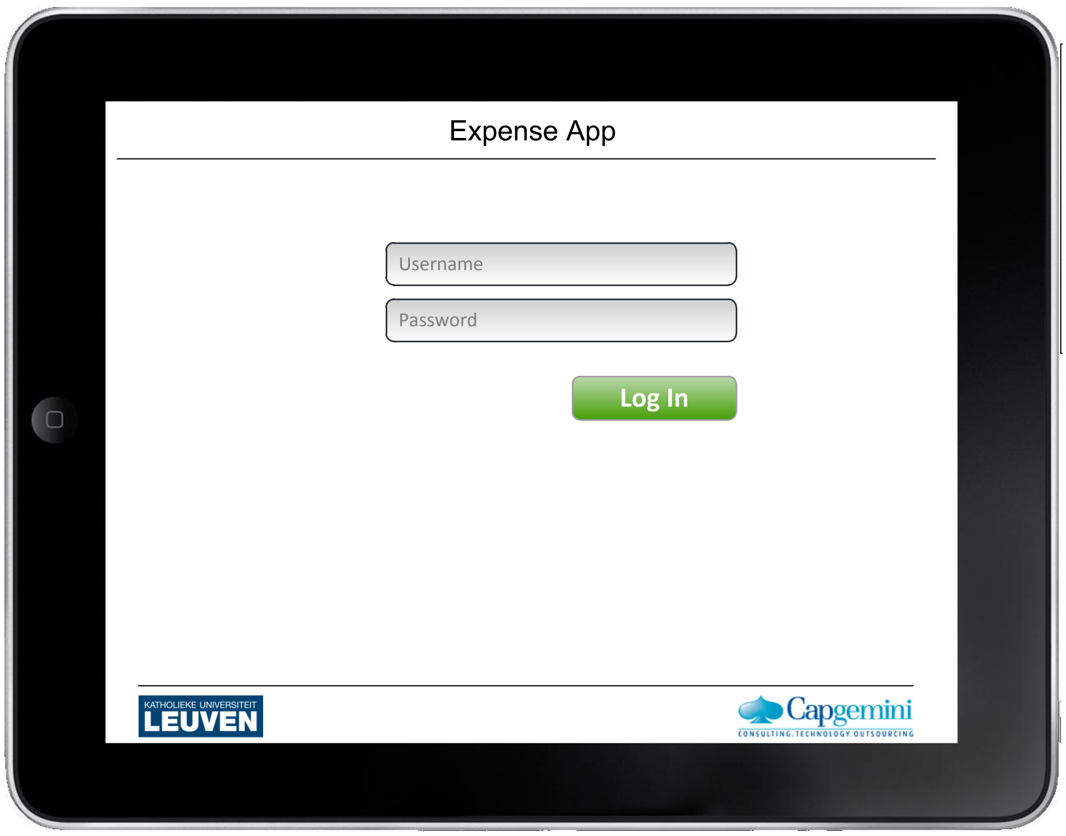
Expense App
Username
Password
Log In
KATHOLIEKE UNIVERSITEIT LEUVEN
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

FIGURE D.1: Login screen

Figure D.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.

D.2.2 Home screen

D.2.3 Wizard: Step 1 (Your Info)

D.2.4 Wizard: Step 2 (Overview)

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

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

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

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

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

D.2.10 Wizard: Validation Dialog

D.2.11 Wizard: Validation element coloring

D.2.12 History

D.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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[15] ——, *Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012*, 2013. [Online]. Available: http://www.gartner.com/newsroom/id/2335616 (visited on 05/01/2013).

[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 (visited on 05/25/2013).

[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 (visited on 11/02/2012).

[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 (visited on 05/01/2013).

[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 (visited on 05/01/2013).

[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` (visited on 04/13/2013).

[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` (visited on 05/26/2013).

[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, *HTML5 vs native: the mobile app debate*, Jun. 2011. [Online]. Available: `http://www.html5rocks.com/en/mobile/nativedebate/` (visited on 02/27/2013).

[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] G. A. Miller, "The magical number seven, plus or minus two", *The Psychological Review*, vol. 63, no. 1, pp. 81–97, 1956. DOI: `10.1037/h0043158.`. [Online]. Available: `http://www.musanim.com/miller1956/`.

[33] T. L. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, ser. Advanced book program. McGraw-Hill, 1980, p. 287, ISBN: 9780070543713.

[34] ——, “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.

[35] L. Zadeh, “Fuzzy sets”, *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965, ISSN: 00199958. DOI: 10.1016/S0019-9958(65)90241-X. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S001999586590241X.

[36] R. E. Bellman and L. A. Zadeh, “Decision-making in a fuzzy environment”, *Management Science*, vol. 17, no. 4, pages, 1970, ISSN: 00251909. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true%5C&db=buh%5C&AN=7123039%5C&site=ehost-live%5C&scope=site.

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

[38] PhoneGap, *PhoneGap Overview*, 2012. [Online]. Available: http://phonegap.com/about/artwork/.

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

[40] Rhodes, *Framework Architecture*, 2012. [Online]. Available: http://docs.rhomobile.com/rhodes/introduction.

[41] B. LeRoux, *PhoneGap Beliefs, Goals, and Philosophy*, 2012. [Online]. Available: http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy/ (visited on 05/18/2013).

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: 681.3

Korte inhoud:

Developing mobile applications (or apps) that support multiple platforms is an expensive and time consuming process. Therefore, many companies are seeking refuge in cross-platform tools (CPT's) 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

Assessoren: ir. Gonzalo Parra
prof. dr. Danny Hughes

Begeleider: ir. Gonzalo Parra