KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

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

Michiel Staessen

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

Promotor:
prof. dr. ir. Erik Duval

Begeleider:
ir. Gonzalo Parra

Academiejaar 2012 – 2013

Preface

Michiel Staessen

Contents

Preface i

Abstract iv

1 Introduction 1

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

1.2 The problem of fragmentation 3
Fragmentation within iOS 4, *Fragmentation within Android* 4

1.3 Cross-platform tools to the rescue 5

2 Literature Study 6

2.1 Multi-Criteria Decision Making (MCDM) 6
Arbitrary scoring models 6, *Analytic Hierarchy Process (AHP)* 7 , *Fuzzy MCDM* 10

2.2 Software evaluation methodology 11

2.3 Mobile Architectures 12
Native App 12, *Web App* 13, *Hybrid App* 14, *Interpreted App* 15, *Cross Compiling* 15

3 Methodology 17

3.1 Define selection criteria 17

3.2 Identify potential candidates 18

3.3 List selected alternatives 18

3.4 Define evaluation criteria 20
CapGemini Interviews 20 , *Literature* 23, *Summary of evaluation criteria* 23

3.5 Evaluate selected alternatives 23
Proof-of-Concept application 24 , *Evaluation methodology* 26

3.6 Select the most suitable alternative 26

4 Selected Tools . . . 27
4.1 Apache Cordova . . . 27
4.2 Motorola Rhodes . . . 27
4.3 Native SDKs? . . . 27
5 Evaluation . . . 28
5.1 Portability . . . 28
Platform support 28 , *Toolset reuse* 28 , *Code reuse* 28 , *Portability Summary?* 29
5.2 Application Experience . . . 29
Native Integration 29 , *UI Capabilities* 29 , *Performance* 29 , *Application Experience Summary?* 29
5.3 Productivity . . . 29
Skill reuse 29 , *Tooling* 30 , *Testing* 30 , *Productivity Summary?* 30
5.4 Global verdict . . . 30
6 Conclusion . . . 31
A Requirements documentation of the proof-of-concept application . . . 33
A.1 Class Diagram . . . 33
A.2 Mock-ups . . . 33
Login screen 33, *Home screen* 34, *Wizard: Step 1 (Your Info)* 34, *Wizard: Step 2 (Overview)* 34, *Wizard: Step 2 (review expense from abroad)* 34, *Wizard: Step 2 (review domestic expense)* 34, *Wizard: Step 3 (add expense domestic expense)* 34, *Wizard: Step 3 (add expense expense from abroad)* 34, *Wizard: Step 4 (Sign & Send)* 34, *Wizard: Validation Dialog* 34, *Wizard: Validation element coloring* 34, *History* 34, *Smartphone version* 34
Bibliography . . . 35

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 presents an overview of the evolution in the mobile device landscape, explains the problem of fragmentation and how cross-platform tools (CPTs) can solve this problem.

1.1 The mobile device landscape

1.1.1 Smartphones

Mobile phones have been around since the nineties and before but the smartphone as we know it now has only been around since the (nearly simultaneous) introduction of the iPhone 3G and the HTC Dream in 2008. In the last five years, smartphone sales have grown tremendously. According to quarterly studies by Gartner[1] [5, 6, 7, 9, 8, 10, 13, 14, 15, 12, 20, 17, 18, 19, 21], 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 has already reached more than 50% [28].

Operating systems are heavily subjected to network effects, i.e. the value of a platform is proportional to the number of people using it. This makes it hard for new platforms to gain traction which is visible in Figure 1.1 and Figure 1.2. Android and iOS are currently the most popular platforms and other platforms are either in decline (like Symbian and Blackberry, formerly RIM) or have a hard time getting traction (like Windows Phone).

However, there is so single major platform. The IDC[2] even predicts that Windows Phone will gain a significant market share by 2016 and that 90% of the worldwide smartphone

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

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

TODO: Update Graph

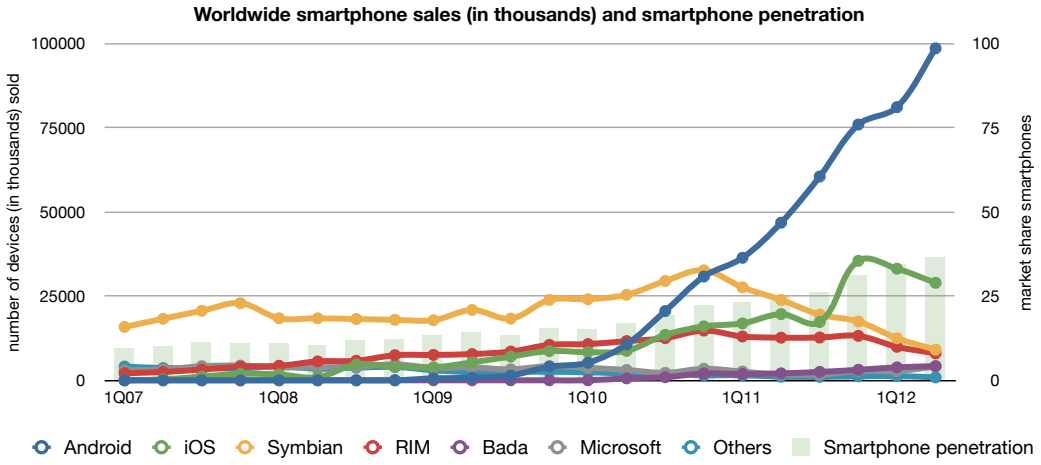

FIGURE 1.1: Growth of worldwide smartphone sales and smartphone penetration. Source: Gartner [5, 6, 7, 9, 8, 10, 13, 14, 15, 12, 20, 17, 18, 19, 21]

TODO: Update Graph

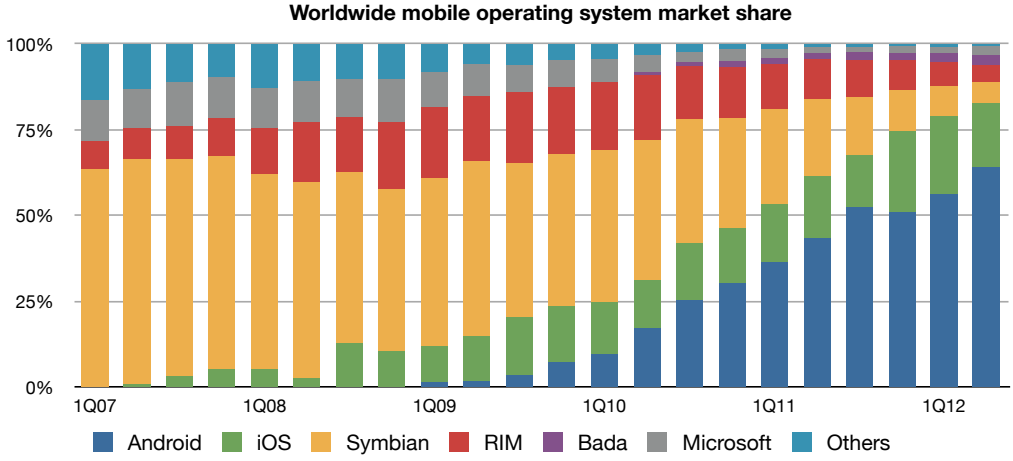

FIGURE 1.2: Growth of worldwide smartphone operating system market share. Source: Gartner [5, 6, 7, 9, 8, 10, 13, 14, 15, 12, 20, 17, 18, 19, 21]

market will then be covered by Android, iOS and Windows Phone [22]. Therefore it is very reasonable to assume that there will always be more than one major platform.

1.1.2 Tablets

A similar scenario is playing out in the tablet industry. According to other studies by both Gartner [11, 16] and the IDC [23], tablets will continue to gain popularity and sales will be mainly driven by iPads and Android tablets (see Figure 1.3).

TODO: Check whether new predictions are available

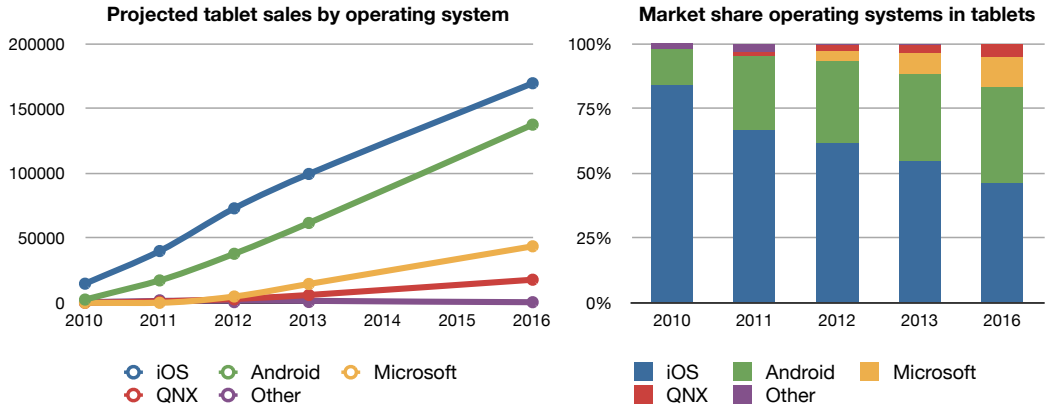

FIGURE 1.3: Prediction of worldwide tablet sales and market share. Source: Gartner [11, 16]

Even though both companies do not agree on which platform will serve the most devices, they both predict there will be at least three major platforms: iOS, Android and Windows.

1.2 The problem of fragmentation

The competition among mobile device manufacturers has led to fragmentation on many levels. For consumers, carriers and manufacturers, fragmentation is usually a good thing. The more different devices there are, the easier it is for consumers to pick the device that fits their needs.

For developers on the other hand, fragmentation is usually a bad thing. They will have to develop and test their applications on multiple devices to be able to guarantee the desired experience. This is expensive and time consuming.

Fragmentation is a multi-dimensional problem. From Figure 1.2 and Figure 1.3 it is already clear that the operating system or platform is one obvious dimension; this is called platform fragmentation. Other dimensions include device configuration, runtime, user interface, etc.

The next subsections will discuss fragmentation issues within iOS and Android.

1.2.1 Fragmentation within iOS

Devices running iOS (commonly referred to as iDevices) are available in different shapes and sizes. Yet, there are only a limited number of (similar) device configurations in two form factors: iPhone (covering iPhone and iPod Touch devices) and iPad. Apple uses these form factors to distinguish three kinds of applications: iPhone apps (designed to run on iPhone form factor only), iPad apps (designed to run on iPad only) and universal apps (designed to run on both form factors).

TODO: describe runtime fragmentation

Overall, Apple can manage fragmentation quite well and fragmentation is rather low within iOS.

1.2.2 Fragmentation within Android

Android is an open source platform, developed by Google. Manufacturers are allowed to tailor it for their devices and many of them have grabbed this opportunity to differentiate their products. On april 24 2013, the Google Play store officially supported 2827 device configurations, distributed across 59 manufacturers. The ability to alter the operating system has largely contributed to the success of Android.

There is however virtually no incentive for device manufacturers to maintain their Android flavours and as a result they do not often provide updates for their devices. This has led to the notorious runtime fragmentation among Android devices (see Figure 1.4).

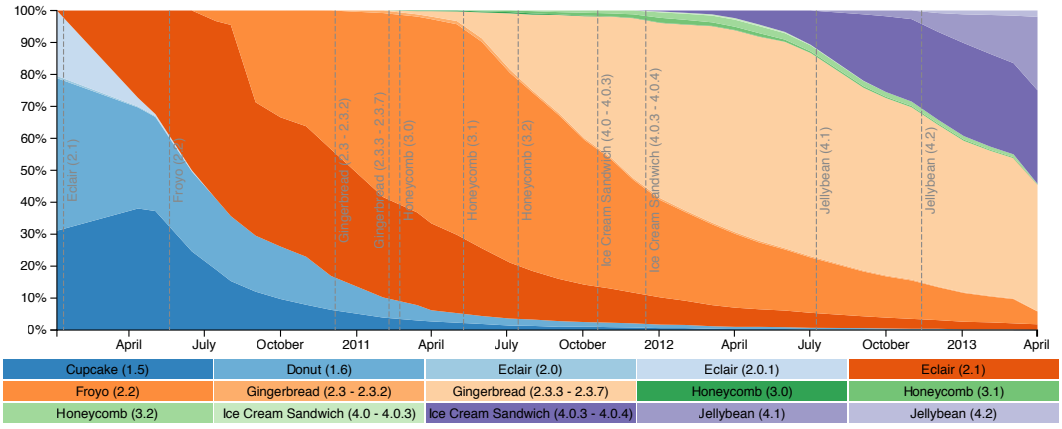

FIGURE 1.4: Historical runtime fragmentation on Android. The data from this graph was aggregated from [1] using the Internet Archive (http://archive.org).

These flavours also contribute to user interface fragmentation because manufacturers ship their devices with a custom user interface to differentiate their product.

Because of the sheer number of Android devices, there is also a large number of CPUs, screen sizes, display resolutions, pixel density configurations and many more. All these differences contribute to device fragmentation.

Long story short, the Android platform is characterized by fragmentation in all dimensions.

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 very unwise to target a single platform. This could eventually lead to lock-in situations which companies try to avoid at all costs. Consequently, they will ask for a cross-platform solution.

Cross-Platform Tools (CPTs) can help solving this problem. They reduce entry barriers (access to new platforms) and exit barriers (lock-in) by allowing developers to create cross-platform applications from a single codebase [26].

Cross-Platform Tools try to solve three major problems [26]:

1. **Fragmentation** The fragmentation issues described above are a pain for every developer. They need 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 can provide workarounds.
2. **Access to new platforms and screens** Targeting a new platform or screen (for instance, television sets or car consoles) is often hard. Developers need to learn yet another SDK and/or programming language in order to deliver applications for the platform or screen. Using a CPT can drastically reduce the effort needed to target a new platform or screen.
3. **Development inefficiency** Maintaining codebases for multiple platforms is a difficult task. When using a CPT, all code is contained within a single codebase and no time is lost while synchronizing features and other maintenance tasks across codebases.

TODO: Finish chapter, which can be read where + define scope, make boundaries

Chapter

2 Literature Study

TODO: write introductory text

2.1 Multi-Criteria Decision Making (MCDM)

Finding the right software package is often a daunting task. In order to suit the end-user's needs, the software should meet a large number of – sometimes conflicting – requirements and will result in making important trade-offs. Because of the these characteristics, software selection can be modeled as a Multiple-Criteria Decision Making (MCDM) problem [24, 25].

There are two categories of MCDM problems: Multiple-Attribute Decision Making (MADM) problems and Multi-Objective Decision Making (MODM) problems. The first category involves sorting and ranking of a limited number of available alternatives, based on a number of decision criteria. In the latter category, there are no alternatives specified beforehand and the number of alternatives is effectively infinite [27].

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

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

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

2.1.1 Arbitrary scoring models

TODO: Question: Is this an appropriate name?

There are a number of arbitrary scoring models but they all have one thing in common: for each criterion, the values are translated to a numerical score. In some cases, this score

can be derived from the value of the criterion itself (e.g. speed, age, cost, ...). In other cases, a mapping is provided (e.g. in order to obtain a score s, at least features f_1, f_2 and f_3 should be supported).

Different methods are available to select potential candidates using these scores [27]:

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

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

The strength of these methods is that they are the most easy to use. However, scores and weights are assigned arbitrarily and might get tough when there are a lot of criteria. Also, not all criteria are suitable for conversion into a numerical scale [24].

2.1.2 Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) is "a multi-criteria decision making approach in which factors are arranged in a hierarchic structure" [30]. It is developed by Thomas L. Saaty and is based on pairwise comparisons of both criteria and alternatives.

The strengths of the AHP are (1) that it enables decision makers to structure a problem into a hierarchy, (2) that it provides a powerful tool for handling both quantitative and qualitative multi-criteria decision making problems and (3) that this system can deal with inconsistency [24].

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

The analytic hierarchy process consists of three stages. In the first stage, the factors that are important for the decision are organized into "a hierarchic structure descending

from an overall goal to criteria, subcriteria and alternatives in successive levels" [30]. Organizing the factors in such structure helps the decision maker in getting an overview of the potentially complex relationships and also helps to assess the relative importance of the issues in each level.

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

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

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

The relative difference between all alternatives with respect to property X can now be summarized in a matrix

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

Note that:

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

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

In reality though, the values w_i are unknown and the ratios $a_{ij} = w_i/w_j$ cannot be calculated. However, an expert could estimate these ratios. The authors have developed a fundamental scale (see Table 2.1), which can be used to rate the difference between alternatives [30].

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

As a result, the aforementioned multiplication would not yield a row of identical values but instead it would yield a row of differing values, scattered around w_i (because of the

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

TABLE 2.1: The fundamental scale for pairwise comparison[30]

errors in judgement). Consequently, it seems reasonable to represent w_i by the average of these values.

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

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

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

which is equivalent to the eigenvalue problem $Aw = \lambda_{max} v$ and its solution is unique to within a multiplicative constant. Small perturbations of a_{ij} could generally lead to large perturbations of λ_{max} and w but it can be shown that this is not the case for reciprocal matrices [29, p. 192 – 197].

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

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

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

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

That is, all n alternatives form a chain.

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

The amount of inconsistency, resulting from human judgement, can therefore be represented by $\lambda_{max} - n$, which measures the deviation of the judgements from the consistent approximation [30].

The authors define a *consistency index*, $C.I. = (\lambda_{max} - n)/(n-1)$. This consistency index is compared to the average consistency index of a large number of random reciprocal matrices of the same order, called the *random index (R.I.)*, listed in Table 2.2. When the *consistency ratio (C.R.)*, the ratio $C.I./R.I.$, is 10% or less, the estimate w is considered acceptable. Otherwise, new judgements should be collected in order to improve consistency.

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

TABLE 2.2: Random indices (R.I.) for reciprocal matrices of order 1 to 9

EXAMPLE TODO: Add example?

2.1.3 Fuzzy MCDM

TODO: Write section

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

Fuzzy MCDM is based on three concepts [**?**]:

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

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

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

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

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

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

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

2.2 Software evaluation methodology

Based on their literature review [24], the authors have proposed a generic, six-stage methodology for the selection of software packages [25].

1. **Define selection criteria.** In the first stage, the evaluator defines the essential requirements for the software. If a certain software package does not meet a selection criterion, it is not a considered to be a suitable candidate and it should not be considered for further evaluation.
2. **Identify potential candidates.** During the next step, the evaluator searches for potential candidates. This step will result in a list of potential candidates.
3. **List selected alternatives.** In this step, the evaluator will use the requirements from stage 1 to filter the list obtained from the previous stage.
4. **Define evaluation criteria.** In this stage, the evaluator has to define the evaluation criteria, arrange them in a hierarchy and define the value scales for each criterion.

5. **Evaluate selected alternatives.** During this phase, the evaluator will make a detailed comparison of the alternatives using the criteria obtained from the previous stage. A methodology like AHP can be used in this stage.

6. **Select the most suitable alternative.** In this final step, all alternatives are ranked using the comparison results from the previous stage. Now, the best alternative can be chosen from the list of candidates. In general, a number of software packages will be taken into consideration and a selection will be made after additional steps (like for instance a cost-benefit analysis and/or contract negotiations with the vendor).

In the original paper [24], the authors also suggest to include an additional evaluation stage after the selected packages has been implemented and integrated. During that stage, one should verify that the selected package does indeed meet the requirements.

2.3 Mobile Architectures

TODO: Explain Mobile Hub/Mobile Orchestrator

TODO: discuss cost for each architecture

There are already a number of paradigms for cross platform mobile application development [4]. This section presents an overview of the available strategies by comparing different aspects: performance, look and feel, platform access, programming languages, development cost and distribution.

2.3.1 Native App

A native app is an application that is specifically designed to run on a particular platform. It is the default approach to develop applications for mobile devices. Figure 2.1 shows an illustration of the overall architecture of such an app.

Native apps are developed with the supplied SDK. Developers will need to get acquainted with the programming language used by said SDK but in return they will get full access to the platform and its features. As a result, the best performance can be obtained with this kind of app.

For the user interface, developers can use lots of interface elements such that they can present a familiar look and feel to the end user.

Native apps can be easily distributed through an online marketplace like for instance the App Store or Google Play.

Because native apps are designed to run on one platform only, this development strategy is not very well suited for cross platform development. If an application should run on multiple platforms, it has to be developed for each platform separately. This is costly.

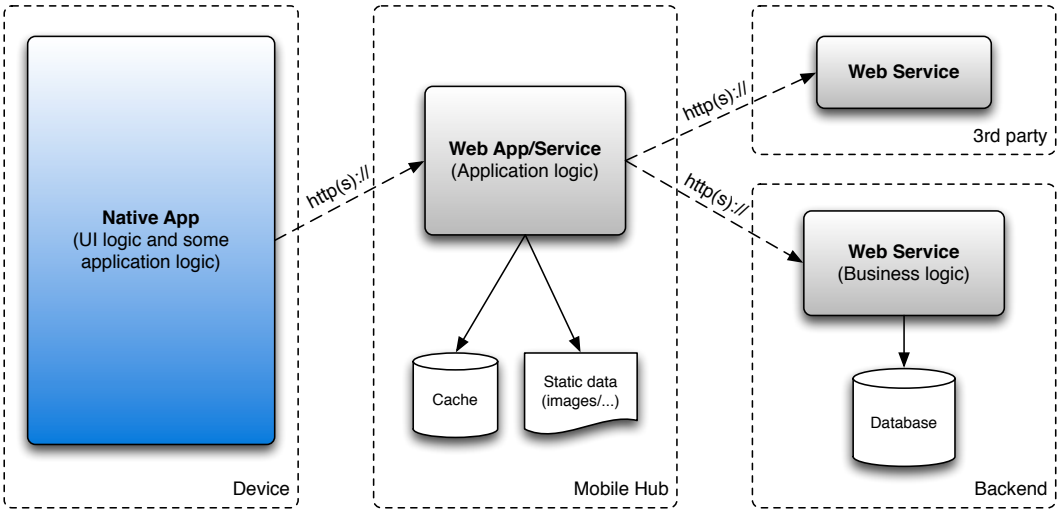

FIGURE 2.1: Overall architecture of a native app.

2.3.2 Web App

Web apps 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 the overall architecture for this kind of app is given in Figure 2.2.

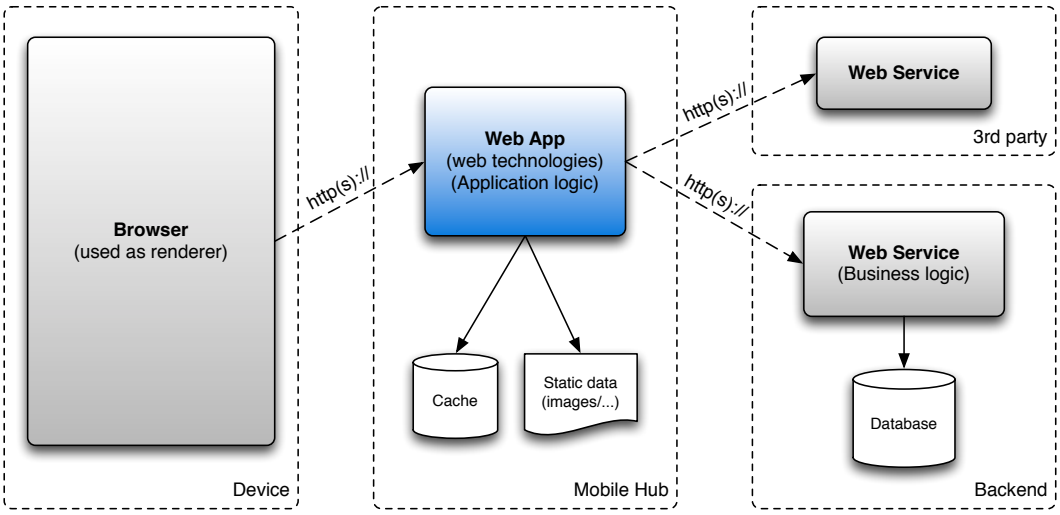

FIGURE 2.2: Overall architecture of a web app.

Web apps are not nearly as powerful as native apps. First of all, the application is not stored on the device. Web apps require an active internet connection which cannot always be guaranteed. Second, they are built with web technologies like HTML, CSS and JavaScript,

which have to be interpreted by the browser at runtime. Third, web apps cannot access the system which means they cannot make use of the many unique features of a mobile device.

With HTML5, web apps can get more powerful. They will be able to access device features, like the camera and other sensors [3]. They will not even require an active internet connection because they can be cached on the device. However, HTML5 is still a draft and a lot of mobile browsers lack proper HTML5 support.

From a user interface perspective, web apps can be a problem as well.

TODO: complete paragraph

Web apps are distributed easily: the only requirement is a valid URL. Web apps cannot be installed on the device though, but there are workarounds using Web Clips on iOS [2] and bookmarks on Android.

2.3.3 Hybrid App

Hybrid applications are the logical next step, combining native apps and web apps. The actual application is a web site, embedded in a web view, part of a native wrapper. The embedded website can access (parts of) the system through a bridge. An overview of the overall architecture is shown in Figure 2.3.

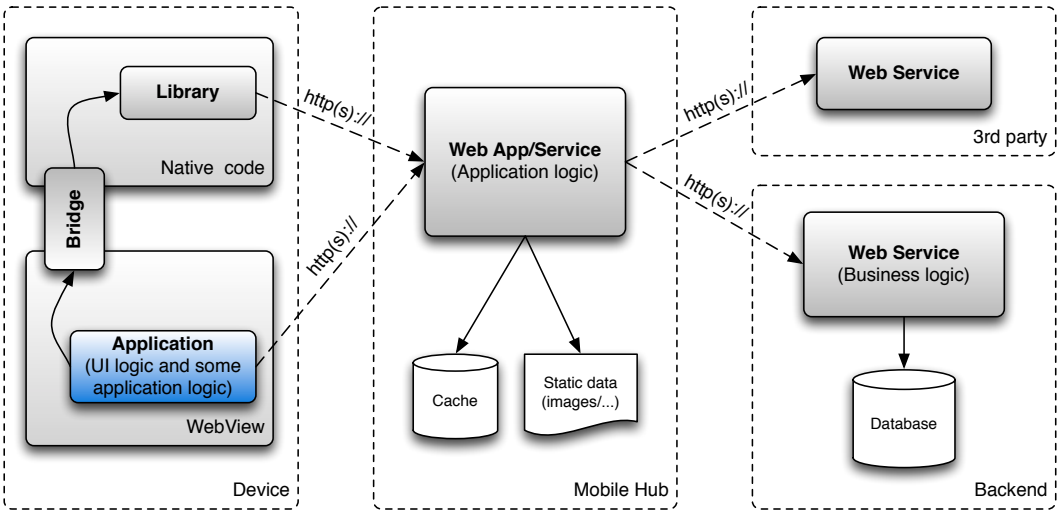

FIGURE 2.3: Overall architecture of a hybrid app.

Hybrid apps are part native app, part web app. Performance will be similar to web apps but some parts can be optimized by using native code. The websites inside the hybrid app are also much more powerful because they can access many device features that aren't available in HTML(5) through the bridge.

When it comes to the user interface, hybrid apps suffer from the same problem as web apps.

Because hybrid apps are wrapped in a native container, they can be distributed just like native applications, through online marketplaces.

2.3.4 Interpreted App

In an interpreted app, instructions in some language are translated to native instructions at runtime. Figure 2.4 shows the overall architecture of an interpreted app.

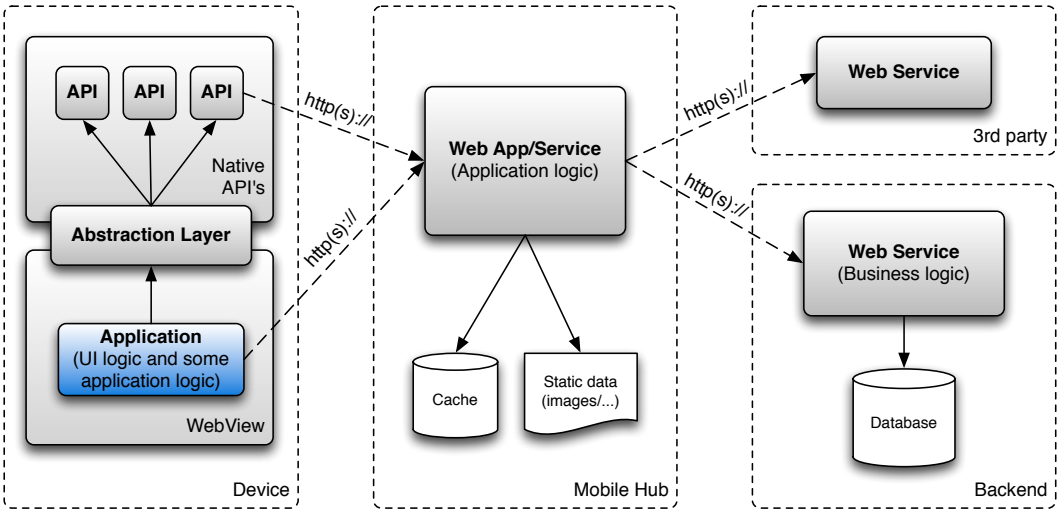

FIGURE 2.4: Overall architecture of an interpreted app.

Performance of interpreted apps depends on the interpreter and interpreted language but is better than web apps on average, though not as good as native apps.

In an interpreted app, the user interface description is interpreted and rendered on the device using native interface elements. An interpreted app will have a familiar look and feel.

From the outside, interpreted apps – just like hybrid apps – look like native apps and can be distributed through online marketplaces.

2.3.5 Cross Compiling

Instead of translating instructions at runtime, one could translate instructions at compile time. The process is called cross compiling and the result is a truly native app. The overall architecture is sketched in Figure 2.5.

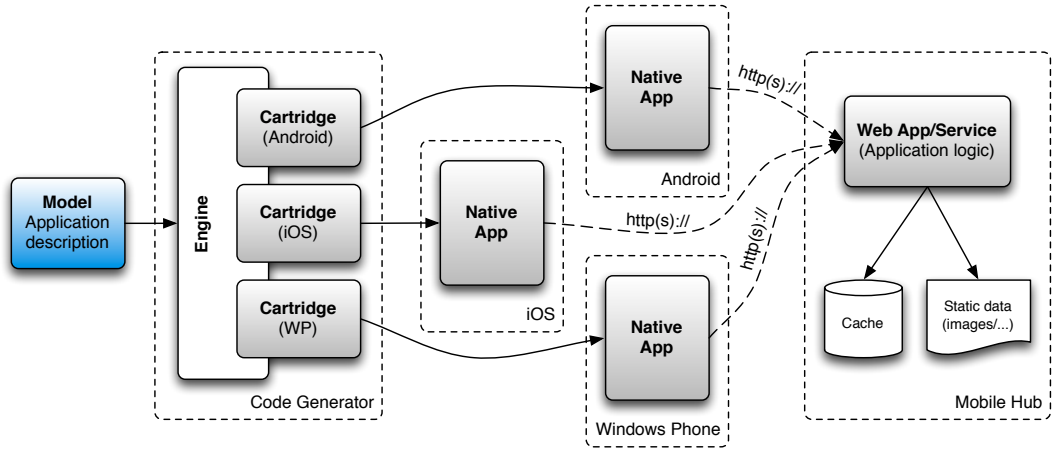

FIGURE 2.5: Overall architecture for cross compiled apps.

Summary

Table 2.3 summarizes the results of the discussed strategies. It is important to note that there is no universal strategy that fits all use cases. A strategy must be chosen carefully, taking into account the client's wishes.

	Native	Web	Hybrid	Interpreted	Cross Compiled
Performance	high	low	rather low	average	high
Platform Access	✓	× / ✓	✓	✓	✓
Look & Feel	native	non-native	non-native	native	native
Distribution	marketplace	URL	marketplace	marketplace	marketplace
Development cost	high	rather low	average	average	average

TABLE 2.3: Summary of cross platform mobile application development strategies.

Summary

Chapter

3 Methodology

This chapter describes the methodology used to compare and rank the studied cross-platform tools. The methodology follows the suggested 6-step approach presented in [25] (see chapter 2.2). The six steps are:

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

Every step will be further expanded in the following sections.

3.1 Define selection criteria

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

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

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

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

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

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

3.2 Identify potential candidates

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

Discovery of cross-platform tools has already been done extensively by VisionMobile. The latest cross-platform tools report [26] contains a list of 100 cross-platform tools they tracked as part of their research. Because additional internet searches did not reveal new tools, this list is used as output of this stage.

3.3 List selected alternatives

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.4 Define evaluation criteria

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

3.4.1 CapGemini Interviews

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

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

Developer

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

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

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

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

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

[1] http://eclipse.org

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

ATTITUDE TOWARDS NEW LANGUAGES AND TOOLS Learning new tools and languages is not considered problematic as long as it is worth investing in and/or requested by clients. Typically, one or more developers familiarize themselves with the tool/language and teach the other developers during a training session.

Mobile architect

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

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

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

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

- **Adoption** How many developers are using this tool? The number of users is mostly a good measure for the quality of the tool. A large user base is also beneficial when facing problems because it is more likely to get answers.
- **Future proof** Will the tool survive? Is there enough financial backing or community effort? It is not very wise to start a project that relies on a tool that has an unclear future.
- **Documentation** Good documentation is priceless as it can save numerous hours of digging through code.
- **Support** In case of a problem, is there an official channel to open a ticket? This could be either paid support or good community support. Either way, it should solve the problem. Delay due to unsolvable problems is unacceptable.
- **Customizability** Clients have specific requirements with regard to user interfaces and the tool should make this possible. Also, can the tool be expanded with plugins? A plugin architecture can contribute substantially to the customizability of the tool.
- **Supported devices** Which devices and runtimes does the tool support? In a B2B[3] context, devices are mostly controlled by an organization and the tool does not need to support many devices. In a B2C[4] context on the other hand, the organization

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

[4] Business-to-consumer, or commerce between a business and consumers

cannot control the devices and consequently, the tool should support a lot more devices. Does the tool cover most or all of the used devices in this case?

- **Performance** The benefits of cross-platform development must justify the costs in terms of performance. If it took virtually no time to develop the application but it is slow, the application is useless.
- **Tools** Does this cross-platform tool allow to reuse the development tools that are already in use? Most people are more productive with the tools they are familiar with.

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

Sales representative

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

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

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

CUSTOMER REQUIREMENTS When clients come to Capgemini with a problem that they would like to solve with a mobile application, they typically do not have special technical requirements. These requirements are most often formulated in other terms. One example is the choice for native versus cross-platform. It could happen that the client has not yet decided on which devices to use or that the client does not want to create a lock-in situation by choosing for a single vendor. In that case, the application clearly has to work cross-platform. Clients could also wish to use NFC technology, which rules out iOS as a platform. When there is no compelling reason to choose for either of both, Capgemini can help the client to make an appropriate decision. Performance is almost never a primary requirement. Perhaps this is implicitly assumed?

[5] Near Field Communication, a wireless technology based on RFID

Summary of interviews

TODO:

3.4.2 Literature

The criteria extracted from the interviews are complemented with criteria from literature; more precisely from reports by VisionMobile and Gartner.

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

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

TABLE 3.2: Top 10 reasons to select a particular cross-platform tool. Respondents could enter their top 3 and only the response for the tools that are selected by 50+ developers are counted, which is 1512 responses [26].

It is clear from Table 3.2 and Table 3.3 that the same reasons are quoted for both selecting and dropping a certain tool.

TODO: Define evaluation criteria from (1) VisionMobile Report and (2) Interviews with CapGemini + PoC

3.4.3 Summary of evaluation criteria

3.5 Evaluate selected alternatives

TODO: Using AHP, compare alternatives. Ask Jan to make 1 score table, derive 1 score table from VisionMobile report. Do the comparison twice or give both scoring tables adequate weights

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

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

TABLE 3.3: Top 10 reasons to drop a particular cross-platform tool. Respondents could enter their top 3 and only the responses for the tools that were dropped by 50+ developers are counted, which is 1109 responses [26].

In this stage, the actual evaluation of the selected tools takes place. The alternatives are compared in pairs, with respect to the criteria recorded in the previous stage. In order to gain the necessary experience that is needed to formulate an accurate judgement, every tool is used to create a proof of concept application, or parts thereof.

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

3.5.1 Proof-of-Concept application

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

Context & scenario

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

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

Typical functional elements

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

- **UI elements** Most applications are useless without a user interface. This application incorporates a number of frequently used UI elements.
  - **Form elements** Virtually all input is captured with "forms". The form elements should support read-write and read-only mode. This application uses different types of form elements to represent different kinds of data.
    - **Text** For inputting arbitrary text.
    - **Number** For inputting numbers.
    - **Email** For inputting email addresses.
    - **Password** For inputting passwords, the content of the input field not shown as characters but as symbols.
    - **Drop-down** For selecting an item from a list.
    - **Radio button** Also for selecting an item from a list.
    - **Toggle switch** To toggle a state on a property, e.g. an on/off switch.
  - **Button**
  - **Tab bar**
  - **Spinner** A spinning wheel that is displayed when the user is supposed to wait while the applications handles some request.
- **UI modes** The application must support multiple screen modes
  - **Tablet UI** The display mode used on tablets. The layout typically consists of a narrow column on the left and a wide column on the right (also known as the master–detail interface).
  - **Phone UI** The display mode used on smartphones. Nearly the same layout but shows less detail master and detail views are decoupled in separate views.
- **serialization** In order to communicate with the backend, data must be serialized using various formats. This application makes use of XML, JSON and plaintext.
- **Input validation**
  - **Data type validation**
  - **Custom validation** When some
- **Sorting**
- **Offline mode**
- **Local Storage**
- **Session management**

3.5.2 Evaluation methodology

3.6 Select the most suitable alternative

TODO: Describe the advantages and disadvantages of the three candidates. Make a decision if only these three candidates are available

Chapter 4

Selected Tools

4.1 Apache Cordova

TODO: Elaborate description of Cordova, including architecture, kind of app, supported platforms, used programming languages, costs, financial backing, community support, official support, ...

4.2 Motorola Rhodes

TODO: Elaborate description of Rhodes, including architecture, kind of app, supported platforms, used programming languages, costs, financial backing, community support, official support, ...

4.3 Native SDKs?

Chapter 5

Evaluation

TODO: Repeat the Methodology sections, this time doing the actual comparison.

5.1 Portability

TODO: High-level portability score

5.1.1 Platform support

ANDROID & IOS TODO: no additional platforms

APACHE CORDOVA TODO: For now: Windows Phone, BlackBerry. For later: Tizen, Qt, Firefox Mobile OS, Ubuntu Mobile, Windows (Desktop)

MOTOROLA RHODES TODO: Windows Phone, BlackBerry

VERDICT TODO: pairwise scores in table

5.1.2 Toolset reuse

ANDROID & IOS TODO: Needed: Mac computer, Xcode, Eclipse

APACHE CORDOVA TODO: Any operating system with a text editor or web IDE, cloud builder available

MOTOROLA RHODES TODO: Any operating system with a Ruby IDE, cloud builder available

VERDICT TODO: pairwise scores in table

5.1.3 Code reuse

ANDROID & IOS TODO: Absolutely nothing!

APACHE CORDOVA TODO: 100%, Except for native plugins: 0%

MOTOROLA RHODES TODO: Any operating system with a Ruby IDE, cloud builder available

VERDICT TODO: pairwise scores in table

5.1.4 Portability Summary?

TODO: summary with all portability values?

5.2 Application Experience

TODO: High-level Application Experience score

5.2.1 Native Integration

ANDROID & IOS TODO: list native device APIs

APACHE CORDOVA TODO: list native device APIs

MOTOROLA RHODES TODO: list native device APIs

VERDICT TODO: pairwise scores in table

5.2.2 UI Capabilities

ANDROID & IOS TODO: list UI elements & styling options

APACHE CORDOVA TODO: list default UI (form) elements in HTML5

MOTOROLA RHODES TODO: same as Cordova minus HTML5 polyfills!

VERDICT TODO: pairwise scores in table

5.2.3 Performance

TODO: subjective testing allowed? AHP can make up for it?

ANDROID & IOS TODO: Best

APACHE CORDOVA TODO: average/worse than average

MOTOROLA RHODES TODO: Worst

VERDICT TODO: pairwise scores in table

5.2.4 Application Experience Summary?

5.3 Productivity

TODO: High-level productivity score

5.3.1 Skill reuse

ANDROID & IOS TODO: Android is Java, iOS is Objective C. Java is known from EE, objective C is not. about 50%

APACHE CORDOVA TODO: Cordova is HTML + JS + CSS, completely known: 100%

MOTOROLA RHODES TODO: Ruby + HTML + JS + CSS, not completely known: 40–50%

VERDICT TODO: pairwise scores in table

5.3.2 Tooling

ANDROID & IOS TODO: Great IDE for iOS, Good Eclipse plugin for Android

APACHE CORDOVA TODO: Few or no tools (Ripple emulator maybe?), Cloud builder?

MOTOROLA RHODES TODO: Awful (out of the box), cloud builder?

VERDICT TODO: pairwise scores in table

5.3.3 Testing

ANDROID & IOS TODO:

APACHE CORDOVA TODO:

MOTOROLA RHODES TODO:

VERDICT TODO: pairwise scores in table

5.3.4 Productivity Summary?

5.4 Global verdict

Chapter

6 Conclusion

Appendices

Appendix

A Requirements documentation of the proof-of-concept application

A.1 Class Diagram

A.2 Mock-ups

A.2.1 Login screen

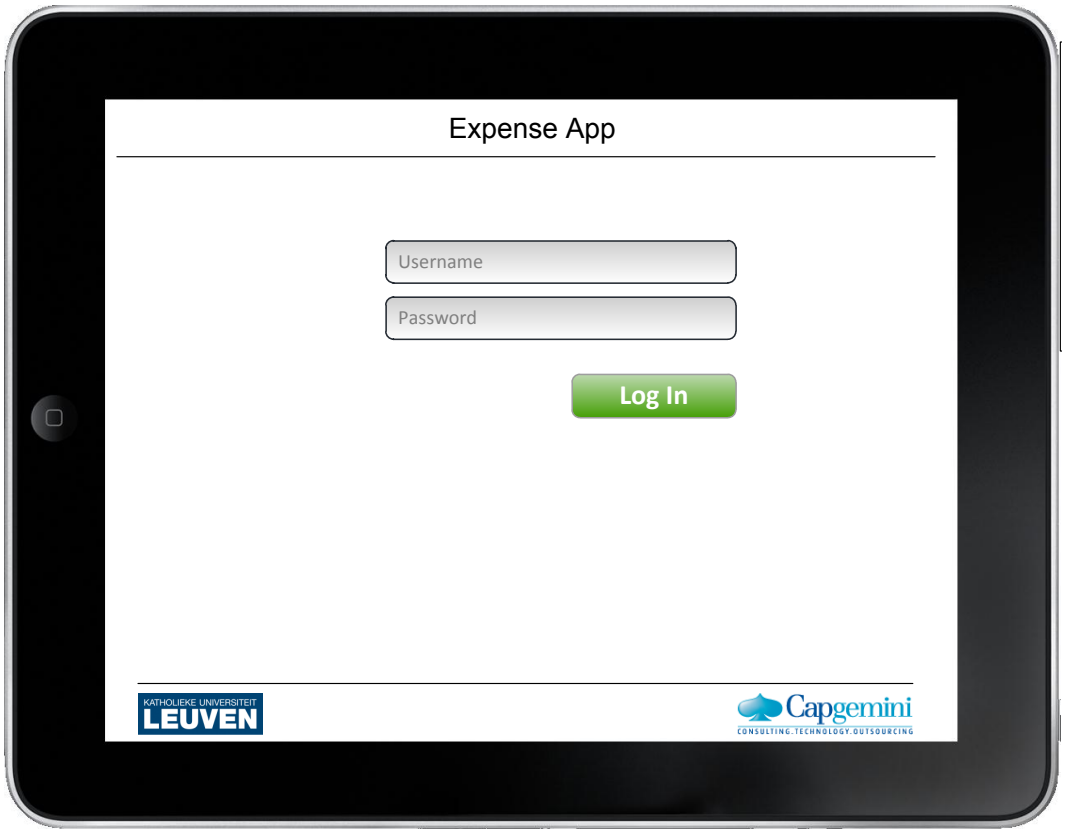

FIGURE A.1: Login screen

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

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

A.2.2 Home screen

A.2.3 Wizard: Step 1 (Your Info)

A.2.4 Wizard: Step 2 (Overview)

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

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

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

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

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

A.2.10 Wizard: Validation Dialog

A.2.11 Wizard: Validation element coloring

A.2.12 History

A.2.13 Smartphone version

Bibliography

[1] Android (2013). Platform Versions, Screen Sizes and Densities and Open GL Version.

[2] Apple (2010). Configuring Web Applications.

[3] Firtman, M. (2012). Mobile HTML5.

[4] Friese, P. (2012). Cross platform mobile development.

[5] Gartner (2008a). Gartner Says Worldwide Smartphone Sales Grew 16 Per Cent in Second Quarter of 2008.

[6] Gartner (2008b). Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 11.5 Per Cent Increase in Third Quarter of 2008.

[7] Gartner (2009). Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008.

[8] Gartner (2010a). Gartner Says Worldwide Mobile Device Sales Grew 13.8 Percent in Second Quarter of 2010, But Competition Drove Prices Down.

[9] Gartner (2010b). Gartner Says Worldwide Mobile Phone Sales Grew 17 Per Cent in First Quarter 2010.

[10] Gartner (2010c). Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent.

[11] Gartner (2011a). Gartner Says Apple iOS to Dominate the Media Tablet Market Through 2015, Owning More Than Half of It for the Next Three Years.

[12] Gartner (2011b). Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent.

[13] Gartner (2011c). Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010.

[14] Gartner (2012a). Gartner Says 428 Million Mobile Communication Devices Sold Worldwide in First Quarter 2011, a 19 Percent Increase Year-on-Year.

[15] Gartner (2012b). Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent.

[16] Gartner (2012c). Gartner Says Worldwide Media Tablets Sales to Reach 119 Million Units in 2012.

[17] Gartner (2012d). 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.

[18] Gartner (2012e). Gartner Says Worldwide Sales of Mobile Phones Declined 2.3 Percent in Second Quarter of 2012.

[19] Gartner (2012f). Gartner Says Worldwide Sales of Mobile Phones Declined 3 Percent in Third Quarter of 2012; Smartphone Sales Increased 47 Percent.

[20] Gartner (2012g). Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth.

[21] Gartner (2013). Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012.

[22] IDC (2012a). Android Expected to Reach Its Peak This Year as Mobile Phone Shipments Slow, According to IDC.

[23] IDC (2012b). IDC Raises Its Worldwide Tablet Forecast on Continued Strong Demand and Forthcoming New Product Launches.

[24] Jadhav, A. S. and Sonar, R. M. (2009). Evaluating and selecting software packages: A review. *Information and Software Technology*, 51(3):555–563.

[25] Jadhav, A. S. and Sonar, R. M. (2011). Framework for evaluation and selection of the software packages: A hybrid knowledge based system approach. *Journal of Systems and Software*, 84(8):1394–1407.

[26] Jones, S., Voskoglou, C., Vakulenko, M., Measom, V., Constantinou, A., and Kapetanakis, M. (2012). Cross-platform developer tools 2012 Bridging the worlds of mobile apps and the web. Technical report.

[27] Kahraman, C. (2008). *Fuzzy Multi-Criteria Decision Making*, volume 16 of *Springer Optimization and Its Applications*. Springer US, Boston, MA.

[28] Nielsen (2012). Smartphones Account for Half of all Mobile Phones, Dominate New Phone Purchases in the US.

[29] Saaty, T. L. (1980). *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Advanced book program. McGraw-Hill.

[30] Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26.

Fiche masterproef

Student: Michiel Staessen

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

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

UDC:

Korte inhoud:

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

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

Promotor: prof. dr. ir. Erik Duval

Assessor:

Begeleider: ir. Gonzalo Parra