Evaluating Cross-Platform Development Approaches for Mobile Applications

Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak

Department of Information Systems, University of Münster, Münster, Germany
{heitkoetter,tima}@ercis.de, sebastianhanschke@gmx.de

Abstract. The fragmented smartphone market with at least five important mobile platforms makes native development of mobile applications (apps) a challenging and costly endeavour. Cross-platform development might alleviate this situation. Several cross-platform approaches have emerged, which we classify in a first step. In order to compare concrete cross-platform solutions, we compiled a set of criteria to assess cross-platform development approaches. Based on these criteria, we evaluated Web apps, apps developed with PhoneGap or Titanium Mobile, and – for comparison – natively developed apps. We present our findings as reference tables and generalize our results. Our criteria have proven to be viable for follow-up evaluations. With regard to the approaches, we found PhoneGap viable if very close resemblance to a native look & feel can be neglected.

Keywords: App, Mobile application, Cross-platform, Multi-platform.

1 Introduction

Smartphones, i.e. mobile phones combining a range of different functions such as media player, camera, and GPS with advanced computing abilities and touchscreens, are enjoying ever-increasing popularity. They enable innovative mobile information systems, often referred to as *apps*. However, the market of mobile operating systems for smartphones is fragmented and rapidly changing. According to Gartner [1], Google's Android, Nokia's Symbian, Apple's iOS, and RIM's Blackberry all have at least a 9 % market share, with Microsoft's Windows Phone expected to increase in popularity as well. The platform distribution among all devices still in use today is even less concentrated. As all platforms differ significantly from each other, software developers that want to reach a large audience of users would be required to develop their apps for each platform separately.

Cross-platform development approaches emerged to address this challenge by allowing developers to implement their apps in one step for a range of platforms, avoiding repetition and increasing productivity. On the one hand, these approaches need to offer suitable generality in order to allow provision of apps for several platforms. On the other hand, they still have to enable developers to capitalize on the specific advantages and possibilities of smartphones.

Our paper first classifies general approaches to cross-platform development of mobile applications. We then analyse and compare existing cross-platform solutions based on Web technologies like HTML, CSS, and JavaScript. As these differ in their general

J. Cordeiro and K.-H. Krempels (Eds.): WEBIST 2012, LNBIP 140, pp. 120–138, 2013.

architecture and their capabilities, it is not obvious which to prefer. We will outline criteria that are important when making a decision as well as evaluate the popular approaches *mobile Web apps*, *PhoneGap* and *Titanium Mobile* according to these criteria.

Our work makes several contributions. Firstly, it gives a comprehensive overview of current approaches for cross-platform app development. Secondly, it proposes a framework of criteria for evaluation. They are not only applicable in this paper but can be used for future assessments. Thirdly, we present a detailed analysis of the considered approaches. Fourthly, we discuss and generalize our findings in order to provide decision advice.

This paper is structured as follows. Related work is studied in Section 2. Section 3 classifies general cross-platform development approaches. Concrete cross-platform frameworks to be evaluated are presented in section 4. We then introduce our catalogue of criteria in Section 5. The evaluation follows in Section 6. In Section 7 we discuss our findings. Eventually, we draw a conclusion in Section 8.

2 Related Work

Much related work can usually be identified for an article that compares various technologies. However, if it deals with cutting-edge technology, the number of similar papers shrinks dramatically. General papers on the technologies dealt with in this paper are cited in the appropriate sections, particularly in Section 4. Thus, this section assesses existing work that compares two or more approaches for cross-platform app development.

Until recently, papers only discussed mobile platforms – or rather operating systems – for mobile devices. An example is the paper by Cho and Jeon [2]. Comparison papers such as by Lin and Ye [3] only marginally help developing multi-platform apps. The same applies to very specialized papers. They usually rather concern the business perspective than deal with technology. An example is a study of mobile service platforms [4]. But even technically-driven papers that address multiple platforms do not necessarily help to develop cross-platform apps. For instance, a study of smartphone malware [5] only roughly hints to platform particularities.

Anvaari and Jansen [6] have compared the predominant mobile platforms with regard to the *openness* of their architectures. Their approach takes a very close look at one aspect and thus can be seen as complementary with our work. Charland and Leroux [7] compare the development of native apps and Web apps. In contrast to our approach, they do not take a cross-platform perspective.

A comparison of iPhone and Android development is presented by Goadrich and Rogers [8]. Despite the topic, which is similar to our work, their aim is different. In fact, they try to answer which platform should be used for the education of students. Another study deals with mobile cloud apps [9]. While the authors deal with cross-platform development, they focus on native thin clients that access cloud services.

A number of publications address more than one platform [10–12]. While these publications foster a better understanding of the platforms, they do not really compare the different approaches. Rather, they explain how to use a technology on a multitude of platforms or devices. Due to the high relevance for practitioners, the topic is also recognized in technology weblogs [13, 14]. Although such articles give valuable advice,

they cannot be compared to our structured approach. In an industry study, VisionMobile compared a large number of cross-platform tools based on a developer survey and vendor interviews. This complements our in-depth review, which is based on a set of criteria.

3 Classification of Approaches

When developing *native applications*, developers implement an application for one specific target platform using its software development kit (SDK) and frameworks. The app is tied to that specific environment. For example, applications for Android are typically programmed in Java, access the platform functionality through Android's frameworks, and render its user interface by employing platform-provided elements. In contrast, applications for iOS use the programming language Objective-C and Apple's frameworks. In case multiple platforms are to be supported by native applications, they have to be developed separately for each platform. This approach is the opposite of the cross-platform idea and will serve as a point of reference in this paper. Users will install native apps from the platform's app store or other platform-provided installation means. They receive an app that, by its very nature, has the look and feel of the platform.

In contrast to separate native development, cross-platform approaches allow developers to implement an app as a single code base that can be executed on more than one platform. We distinguish between approaches that employ a runtime environment and those that generate platform-specific apps from a common code base at compile time. The latter, generator-based category includes model-driven solutions and cross-compiling. Up to now, there are no production-ready solutions of this category. Hence, we concentrate on cross-platform solutions that combine the source code of an app with a runtime environment. This environment interprets the app's code at runtime and thereby executes the app. The runtime environment has to be specific for each mobile platform, while the app's source code is platform-independent. Three different kinds of environment can be identified: the Web browser, a hybrid of Web and native components, and self-contained environments.

Mobile Web applications (*Web apps*) implemented with HTML, CSS, and JavaScript use the browser as their runtime environment and thereby capitalize on the good browser support of mobile platforms. When using this approach, developers implement their application as *one* Web site optimized for mobile devices, which the Web browser then interprets. The optimization has to account for the different screen size and usage philosophy of mobile devices. Due to the standardized technologies, the Web site can be accessed in a similar way by mobile browsers on all platforms. However, mobile Web apps cannot use device-specific hardware features such as camera or GPS sensor. They usually cannot be installed on the mobile device but are retrieved via an URL. The upcoming version of HTML, HTML5, will provide some means in both areas, but not a comprehensive solution. Typically, Web apps will at least partially look and behave like common Web pages, differing in appearance and behavior from the native UI elements provided by the platform.

To resolve the lack of access to hardware functionality while still satisfying the desire to employ common Web technologies, *hybrid approaches* emerged as a combination of Web technologies and native functionality. Their runtime environment largely consists of a Web rendering engine, wrapped in a native engine. The source code of hybrid apps uses similar technology like Web apps but additionally has access to an API for platform-specific features. At runtime, the platform's Web view—essentially a browser without user controls—interprets the source code to display Web pages. All calls to hardware APIs are relegated to the native wrapper. Hybrid apps are packaged natively and thus can be (and have to be) installed on the device, unlike Web apps. While their look & feel mostly resembles that of Web apps, they have access to platform-specific features.

Self-contained runtime environments do not reuse any (Web) environment already present on mobile platforms, but use their own, separate runtime environment. Since such an engine is built from scratch and not based on any previous engine, building a self-contained environment needs more work, but also offers more freedom. Self-contained frameworks are not constrained by existing environments and can be designed according to the needs of apps. Hence, they can enable an intuitive and easy development process. Apps are typically packaged with the framework's engine and deployed as native packages.

4 Overview of Frameworks

Based on the classification from above, we chose three concrete cross-platform solutions, i.e. frameworks, and evaluated them, one from each kind of runtime environment: mobile Web apps, PhoneGap as a hybrid framework, and Titanium Mobile as a self-contained approach. On the one hand, their evaluation is useful on its own, because these are popular frameworks among developers[1]. On the other hand, each also stands as an example of their category, so that their evaluation offers additional insight into the general suitability of each category. Together, they make up the largest part of the decision space relevant when thinking about cross-platform development for mobile devices. Native apps will serve as a point of comparison. The following briefly introduces each framework.

Simple Web apps may rely on the browser itself and do not necessarily need to be supported by a concrete framework. However, several frameworks support the optimization for mobile use, e.g. jQuery Mobile [17] or Sencha Touch [18]. Our evaluation mainly applies to Web apps in general, although our tests have used jQuery Mobile as a concrete framework.

The most prominent exponent of the hybrid approach is *PhoneGap* [19]. PhoneGap was originally created by Nitobi Software, which has been acquired by Adobe [15]. The development now takes place in the Apache Cordova project of the Apache Foundation [20], of which PhoneGap is a distribution [21]. There, it is developed as open source under Nitobi's leadership by a diverse community, including developers from major

[1] PhoneGap counts more than half a million downloads and thousands of applications built with the framework [15]. Numbers from Appcelerator indicate 30,000 applications using Titanium [16].

software firms like IBM or Microsoft [22]. Using PhoneGap, developers still implement their application with HTML, CSS, and JavaScript. In addition to the Web view, PhoneGap's runtime environment provides a JavaScript API combined with a native bridge to access hardware features. We did our initial review with PhoneGap 1.2, using jQuery Mobile 1.0 for mobile optimization. Where necessary, we have since updated our evaluation to version 1.8.0 of PhoneGap and jQuery Mobile 1.1.0.

As a self-contained runtime environment, Appcelerator *Titanium Mobile* [23] follows a different approach. It does not use HTML and CSS to create the user interface. Instead, the UI is implemented completely programmatically. Developers use JavaScript to build the interface and to implement logic and data, extensively using the Titanium API. The code is then packaged with Titanium's engine. At runtime, this engine interprets the JavaScript code and creates the user interface. Similar to PhoneGap, apps can then be distributed via app stores. However, their look-and-feel resembles the typical platform appearance more closely; the UI is made up of native elements. Titanium Mobile is a product of Appcelerator, which leads development of the basic platform provided as open source [24] and sells additional features and support. We did our initial tests with Titanium Mobile 1.7.2 and have since updated our review to version 2.0.2.

Other frameworks not covered here are, for example, Rhodes [25], a hybrid approach similar to PhoneGap, and model-driven approaches. The latter category has not been included because existing model-driven solutions like iPhonical [26] or applause [27] are still in early stages or not relevant in general practice. The same applies to cross-compiling tools like XMLVM [28].

5 Criteria

In the following, we will elaborate on a list of criteria for evaluating cross-platform development approaches. In Section 6, this set of criteria will be used to compare and review the solutions outlined in the previous section. The selection of these criteria is based on and has been influenced by various sources. An initial set of criteria emerged from discussions with practitioners and domain experts from small- to medium-sized software firms. They outlined their requirements for mobile development approaches. These have been augmented through literature research [29–31] and a compilation of typical problems apparent in online developer communities. Furthermore, important experiences regarding necessary features have been gained from developing prototypical apps.

For a better overview, the consolidated list of 14 criteria has been structured into *infrastructure* and *development* perspective. The infrastructure perspective sums up criteria relating to the life-cycle of an app, its usage, operation and functionality/functional range (see Table 1). The development perspective covers all criteria that are directly related to the development process of the app, e.g. topics like testing, debugging and development tools (see Table 2).

Table 1. Criteria of the infrastructure perspective

I1 **License and Costs** This criterion examines whether the framework in question is distributed as free software or even open source, the license under which it is published, if a developer is free to create commercial software, and whether costs for support inquiries occur.
I2 **Supported Platforms** Considers the number and importance of supported mobile platforms, with a special focus on whether the solution supports the platforms equally well.
I3 **Access to platform-specific features** Includes access to device hardware like camera or GPS and to platform functionality like contacts or notifications. Compared according to application programming interface (API) and Web site.
I4 **Long-term feasibility** Especially for smaller companies the decision for a framework might be strategic due to the significant initial investment. Indicators for long-term feasibility are short update cycles, regular bug-fixes, support of newest versions of mobile operating systems, an active community with many developers, and several commercial supporters steadily contributing to the framework's development.
I5 **Look and feel** While the general appearance of an app can be influenced during development, it does matter whether a framework inherently supports a native look & feel or whether its user interface looks and behaves like a Web site. Most users seek apps that resemble native apps. Furthermore, this criterion tries to ascertain how far a framework supports the special usage philosophy and lifecycle inherent to an app. Apps are frequently used for a short amount of time, have to be "instant on", and are likely to be interrupted, e.g. by a call. When returning to the app, a user does not want to repeat her input but wants to continue where she left the app.
I6 **Application Speed** Tries to compare the application's speed at start-up and runtime, i.e. its responsiveness on user-interaction. For evaluation, instead of measuring the performance, we assess the subjective user-experience.
I7 **Distribution** Evaluates how easy it is to distribute apps created with the respective framework to consumers. One part is the possibility to use the app stores of mobile platforms, since users often want to use this distribution channel. However, solely relying on app stores also has disadvantages; a framework offering additional channels also has merits. Furthermore, this criterion assesses whether updates are possible.

6 Evaluation

We have evaluated the four solutions described in Section 4 according to the criteria of Section 5. The evaluation draws on an analysis of the solutions informed by own research and experiences as well as opinions from experienced developers. The experience was mainly gathered by developing a prototypical task management app employing all four solutions. Typical problems arising when using these solutions were compiled from observing the respective developer communities and completed the background information for the evaluation. In addition to a textual evaluation, we assessed a solution's fulfillment of each criterion on a scale from 1 to 6, with 1 meaning "very

Table 2. Criteria of the development perspective

D1 **Development environment** Evaluates maturity and features of the development environment typically associated with the framework, particularly the tool support (IDE, debugger, emulator) and functionalities like auto-completion or automated testing. The term "ease of installation" summarizes the effort for setting up a fully usable development environment for a framework and a desired platform.
D2 **GUI Design** This criterion covers the process of creating the graphical user interface (GUI), especially its software-support. A separate WYSIWYG editor and the possibility to develop and test the user interface without having to constantly "deploy" it to a device or an emulator are seen as beneficial.
D3 **Ease of development** This criterion sums up the quality of documentation and the learning-curve. Therefore, the quality of the API and documentation is evaluated. This part of the criterion is well-fulfilled if code examples, links to similar problems, user-comments, etc. are available. The learning curve describes the subjective progress of a developer during his first examination of a framework. Intuitive concepts bearing resemblance to already known paradigms allow for fast success. This can have a significant impact on how fast new colleagues can be trained and how much additional, framework-specific knowledge a developer needs to acquire.
D4 **Maintainability** The lines of code (LOC) indicator is employed to evaluate maintainability [32, p. 53f.]. The choice of this indicator is based on the assumption that an application is easier to support when it has less LOC, because e.g. training of new developers will be shorter, source code is easier to read etc. While more sophisticated approaches could also be justified as relevant indicators, these are hard to apply, especially in case of complex frameworks and for apps composed of different programming and markup languages.
D5 **Scalability** Scalability is based on how well larger developer teams and projects can be conducted using the respective framework. Modularization of framework and app are highly important as this allows increasing the number of concurrent developers and the scope of the app's functionality.
D6 **Opportunities for further development** Determines the reusability of source code across approaches and thereby assesses the risk of lock-in, which would be increased if a project started with one framework could not later be transferred to another approach.
D7 **Speed and Cost of Development** Evaluates the speed of the development process and factors that hinder a fast and straightforward development. Costs are not explicitly estimated because they are taken as being dependent on the speed of development, assuming that one can abstract from differences in salary of a JavaScript or Java developer.

good" and 6 "very poor". This allows for a quick overview. Due to space restrictions we present the results in tabular form, with two tables per solution, one for the infrastructure and one for the development criteria, and summarize the main findings for each solution in the following subsections. Section 7 draws a comparison between the solutions and provides decision support.

6.1 Web App

Table 3 and Table 4 present the evaluation of mobile Web apps as a cross-platform development approach. Web apps can be accessed from all smartphones via the platform's browser. They are based on open and mature standards and enable easy and fast development. The disadvantage of this approach is its lack of hardware access and that the look and feel resembles Web sites. While Web apps can easily be accessed via their URL, it is not possible to use the distribution and marketing facilities of app stores. This limits their feasibility for commercial applications.

6.2 PhoneGap

Table 5 and Table 6 present the evaluation of PhoneGap as a hybrid cross-platform development approach. PhoneGap offers generic access to platform-specific features on all major mobile platforms. Because it is based on Web technology, development is only slightly more complicated compared to Web apps. However, as a consequence, the visual appearance and, to a lesser extent, the behavior do not reflect a native look and feel but rather that of a Web site.

6.3 Titanium Mobile

Table 7 and Table 8 present the evaluation of Titanium Mobile as a cross-platform approach based on a self-contained runtime environment. As its main advantage, apps built with Titanium Mobile inherently have the look and feel of native apps, although performance limitations might impair the user experience in certain situations. Titanium only supports iOS and Android; the entire ecosystem is less *open*. Advanced features often require a subscription. Developing apps with Titanium requires a high amount of Titanium-specific knowledge, which, together with the programmatic GUI creation, slows down development.

6.4 Native App Development

Table 9 and Table 10 present the evaluation of native development for Android and iOS. Apps developed specifically for each platform using their APIs and following their conventions inherently results in a native look and feel. However, this has to be done separately for each platform and thus does not represent a cross-platform development approach. Abstracting the results from the concrete platforms it can be said that native development benefits from the best support but requires very specific knowledge.

7 Discussion

This section offers a synthesis of the evaluation and provides general advice for choosing a suitable cross-platform approach. Although native apps benefit from an optimal integration into the respective mobile operating system and good developer support, the analysis showed that cross-platform approaches are a viable alternative. As soon as mobile apps have to be developed for multiple platforms under tight budgets, with small

Table 3. Evaluation of mobile Web applications – Infrastructure perspective

I1 **License and Costs** Fees may apply for using specific JavaScript frameworks. Although most of these are open-source [17, 18], there are some examples that require commercial licenses [33]. Most communities are very active and usually answer questions in community boards, which might be seen as *free* support. Nevertheless, selling support packages is a typical business model for open-source software. Moreover, costs may occur from hosting (storage and traffic) a Web site.	3
I2 **Supported Platforms** All smartphone platforms have their own native browser. Additionally, there are several alternatives, e.g. Mozilla Firefox or Opera Mini. Hence, support of the different platforms only differs in browser quality. Most native browsers use the WebKit library, but there are minor variations in displaying the user interface [34].	1
I3 **Access to platform-specific features** JavaScript does not permit any hardware access on smartphones. HTML5 offers "WebStorage" to locally store application data. This concept, however, is in most browsers limited to 5 MB [35]. Playback of video and audio files and the use of multi-touch gestures are no longer a problem.	5
I4 **Long-term feasibility** HTML, CSS, and JavaScript are well established techniques undergoing steady improvement. The decision for a specific JavaScript framework can however turn out to be problematic because changing it later on is in most cases expensive. Nevertheless, there are some popular and widespread frameworks that can be assumed future-proof due to a very active development, regular bug-fixes, and a large community.	1
I5 **Look and feel** The usage of native UI elements from within the browser is not possible; design and layout of apps depend on CSS. There are several projects trying to imitate the design of a specific platform, e.g. *CSS Theme for iPhone* [36]. jQuery Mobile does not follow this approach and manual work is necessary. CSS3 facilitates simple and fast development of user interfaces. There are major differences in the usage philosophy of a Web site and an app. The browser can be closed at any time and does not have to notify the Web site of this event. Whenever the users returns to a Web app, the app should have memorized settings and input, which, thanks to HTML5, has become possible. By using a manifest file [37], a Web site can request to keep an offline copy, concepts like WebStorage allow Web sites to save data in the local storage.	4
I6 **Application Speed** Due to the fact that a Web app has to be loaded via the Internet, launching the app may be slow. WebStorage and the manifest file (as described in I5) limit this phenomenon to the first start of an app. This is comparable to the installation of a native app from an app store. At runtime, Web apps profit from the fact that today's smartphone browsers are highly performance-optimized. Still, the authors' experiments with this approach have shown that especially with a high number of animations and large amounts of content an app can easily reach the limit of a smartphone's CPU.	3
I7 **Distribution** Distributing a Web app is simple. Users only need to know its URL and they will automatically get the most recent version. Using app stores is generally not possible. One could package the Web app via PhoneGap or Titanium; however, this is not permitted in Apple's app store as there is no additional benefit of doing so [38].	3

Table 4. Evaluation of mobile Web applications – Development perspective

D1 **Development environment** There are several development environments for developing with HTML, CSS and JavaScript. They provide almost all desired functionality such as auto-completion. Installing the software development kit (SDK) of the desired platform is mandatory for the use of an emulator, although, for a first impression, a desktop-browser might be enough. In summary, the maturity of development tools is high. Software support for debugging and testing is excellent; in most cases tools like *Firebug* [39] can be employed in addition to a regular browser.	2
D2 **GUI Design** Most tools for Web UI design offer WYSIWYG editors. These need to have special settings for e.g. display size and resolution to be helpful when developing smartphone apps. As the Web app can rapidly be reloaded on the target device without having to recompile it, GUI design is comparably fast.	1
D3 **Ease of development** As the quality of documentation (again depending on the framework used) is very high and as concepts used in HTML, CSS and JavaScript are intuitive, the ease of development is higher than with any of the other frameworks. Besides having to know the underlying programming and markup languages (HTML, CSS, and JavaScript), a programmer does hardly need any further knowledge. He has to be aware of characteristics and limitations of a smartphone (display size, Web storage, limited CPU and GPU speed [40]) and can then start developing.	2
D4 **Maintainability** A *good* JavaScript framework enables short and elegant code. Functionality like sorting of data can sometimes be inserted by using a single keyword. The underlying framework will then supply all necessary methods. The LOC indicator for the prototype application was lowest for the mobile Web application.	1
D5 **Scalability** Web apps in general can easily be split into a large number of small files that fit into the overall design. This might again depend on the framework employed. Projects using jQuery, for example, tend to become confusing from a certain size [41] while others support modularization very well.	2
D6 **Opportunities for further development** A project started as a Web app can easily be ported to PhoneGap if access to the native API should become necessary. It might also be packaged with a WebView control in Titanium Mobile or as a native application, although both would contradict the "native" character of these apps and not provide all of the advantages of these approaches. Altogether, opportunities for further development are excellent.	1
D7 **Speed and Cost of Development** In comparison to all other frameworks, developing the prototype as a Web app has taken the shortest amount of time. Development tools are technically mature, debugging and testing and the design of the user interface can therefore be carried out fast and cost-efficient.	1

developer teams, and in a short time frame, a cross-platform approach is necessary. However, these approaches are more than a second-best alternative. Developers might prefer using a cross-platform solution even in the absence of these constraints.

Mobile Web apps constitute an ideal starting point for cross-platform, because they do not require advanced knowledge and enable developers to start implementing the app right away. Web apps are a simple approach benefiting from good support by

Table 5. Evaluation of PhoneGap – Infrastructure perspective

I1 **License and Costs** Both PhoneGap and jQuery Mobile are open source software (distributed under Apache License 2.0 [42], respectively GPL/MIT license [43]). Commercial software can be created free of charge. Nitobi sells support packages from USD 25 to $\geq$ USD 2000 per month, including bug fixes and private telephone support [44].	2
I2 **Supported Platforms** PhoneGap supports seven mobile platforms (iOS, Android, BlackBerry OS, Windows Phone, HP WebOS, Symbian, Bada); this is only beaten by Web apps. The amount of supported features differs slightly, even among different versions of the same operating system. As PhoneGap uses a platform's Web view, JavaScript frameworks that are intended to be used in addition to PhoneGap need to support each targeted platform. jQuery Mobile supports all platforms for which PhoneGap is available [45].	2
I3 **Access to platform-specific features** PhoneGap gives easy access to most platform-specific features [46]. More sophisticated functionality, e.g. scanning of barcodes, can be added via plugins.	2
I4 **Long-term feasibility** As both PhoneGap and jQuery Mobile are comparatively young projects, with their first version released in August 2008 respectively October 2010, long-term feasibility is hard to estimate. Adobe acquiring Nitobi [15], support from IBM [22], becoming an Apache project [20], and regular bug fixes and updates all are in favor of PhoneGap. The same can be said about the active community, which developed numerous plugins and offers support on community boards. This also applies to jQuery Mobile.	2
I5 **Look and feel** In contrast to apps developed natively, PhoneGap does not use native user interface elements. Using CSS to imitate the native appearance of a platform requires a high amount of manual work. jQuery Mobile's standard stylesheet tries to imitate the iOS look and feel, but differences remain noticeable. The life-cycle of an app is far better implemented in PhoneGap than it is in Web apps. PhoneGap offers events that are triggered for all relevant changes in an app's status, e.g. pause or resume.	3
I6 **Application Speed** Launching a PhoneGap app is fast and user interaction is smooth. Even many tasks did not influence the prototype's performance, which is comparable to a native app.	1
I7 **Distribution** Although Apple reserves its right to decline apps that are primarily Web apps, this does not apply to apps developed with PhoneGap, insofar its API is used to access hardware or platform-specific features [47]. Hence, PhoneGap apps and updates can in general be distributed via app stores.	2

mobile browsers on all platforms. Furthermore, they can be easily ported to other cross-platform approaches.

As soon as platform-specific functionality not available from within browsers has to be accessed or if distribution via app stores is deemed important, other approaches are necessary. Both PhoneGap and Titanium Mobile fulfill these requirements. Their main difference lies with the look & feel of apps developed with these approaches. If it is a strict requirement that an app's user interface should appear like a native app, Titanium is to be preferred. However, Web apps or apps built with PhoneGap merely tend to look slightly different from native apps and more like Web sites, which might

Table 6. Evaluation of PhoneGap – Development perspective

D1 **Development environment** As is the case with Web apps, the developer is not limited in his choice of a development environment when using PhoneGap. However, not all IDEs offer auto-completion for PhoneGap's API. *PhoneGap Build* is a service that compiles an app for different platforms in the cloud, so that developers do not have to install the platform SDKs [48]. After providing the source of a PhoneGap app, apps are compiled and signed for all chosen platforms and can easily be downloaded.	2
D2 **GUI Design** As for Web apps, designing the graphical user interface can largely be done using a standard browser and WYSIWYG editors like Adobe Dreamweaver.	1
D3 **Ease of development** PhoneGap's documentation is clearly structured and comprehensive [49]. It provides numerous examples – in most cases one quick and one full example – and in some cases mentions problems with specific methods on a certain platform. The documentation of jQuery Mobile is equally good [50]. Almost no further knowledge is required in addition to these APIs. The last releases of PhoneGap had some stability problems, which have, however, been fixed by now [51].	2
D4 **Maintainability** Except for additional code that accesses the hardware, hybrid apps do not require more lines of code than comparable Web apps. Implementing our prototype with PhoneGap, we got the impression that the source code is short and clearly structured, largely due to the use of jQuery Mobile.	1
D5 **Scalability** The evaluation of Web apps with respect to this criterion applies without modification.	2
D6 **Opportunities for further development** A project using PhoneGap can, as long as no platform-specific features are used, also be run as a mobile Web site. This enables a company to reach even those customers that do not own a smartphone with an operating system supported by PhoneGap or that do not want to download and install an app.	2
D7 **Speed and Cost of Development** This is more or less equal to those of a Web app, with only little additional time required for implementing access to hardware functionality.	1

even be desirable. This should be kept in mind before postulating native look & feel as a must-have, especially as the look & feel criterion (I5) is the only one where Titanium performs better than PhoneGap. The main disadvantages of Titanium are that it supports only two platforms – albeit the most important ones –, its less open business model, and a more complicated development process. Thus, if there are no hard requirements regarding look & feel or if these might be loosened, the evaluation showed PhoneGap to be the preferable option for cross-platform development.

However, these are only general guidelines that have to be adapted and interpreted for each project individually. The results of our evaluation can be used to support such decisions, for example in semi-formal multi-criteria decision methods like the weighted sum model [64]. Basic decision support can be obtained by weighing the criteria according to the requirements of a given project and calculating a weighted grade. Carefully interpreted and analysed for sensitivity, the result might yield first insights on which solution best matches the requirements at hand.

Table 7. Evaluation of Titanium – Infrastructure perspective

I1 **License and Costs** While Appcelerator provides a community edition of Titanium Mobile free of charge and as open source, this edition is limited in functionality. Additional functionality is available in proprietary, closed-source modules, which are only available with a subscription [52]. Subscription packages include support, while basic documentation is available in Appcelerator's *developer center*. In general, the Titanium ecosystem is less open than the other solutions.	5
I2 **Supported Platforms** As of June 2012, Titanium supports iOS and Android, with Android being slightly less well supported. Consequently, a large number of API methods are "iOS only". While this enables developers to use the latest iOS API, it harms cross-platform compatibility, as platform-specific code might be necessary in certain circumstances. Version 2.0.1 of Titanium introduced the possibility to also generate mobile Web apps. Since this "Mobile Web platform" is still in development and will not support platform-specific APIs [53], we do not consider it further.	4
I3 **Access to platform-specific features** Titanium's spectrum of functionality can be compared to that of PhoneGap [54].	2
I4 **Long-term feasibility** Appcelerator's Web site explicitly mentions its large community with numerous developers and projects. Nevertheless, the community seems to be less active than PhoneGap's. Some posts in Appcelerator's bulletin board remain unanswered for weeks. This might be explained by the comparatively less open nature of Appcelerator. Appcelerator tries to embed current trends into their framework, e.g. using latest functionality of the operating systems. Updates and bug-fixes occur continuously. However, as Titanium Mobile is driven by a single company, the long-term outlook depends largely on their corporate strategy.	3
I5 **Look and feel** Instead of using HTML5 and CSS3, Titanium interprets an app's JavaScript code by creating native UI elements for the app's user interface [55]. At first sight this approach seems to be less intuitive. Even drawing a label or a button requires relatively much knowledge about Titanium's JavaScript API. Ultimately, creating a user interface that resembles a native app requires far less time and effort than with Web apps or PhoneGap. The usage lifecycle of an app can easily be implemented.	2
I6 **Application Speed** At start-up, the Titanium prototype did not differ from those created with other frameworks. At runtime, it started to noticeable stutter as soon as many objects and thus a large amount of view elements had to be handled. As the prototype is rather simple, programming errors can quite certainly be ruled out. It is more likely that this stems from the interaction of operating system and Titanium's JavaScript interpreter.	5
I7 **Distribution** Titanium apps can be distributed via the different app stores without difficulty.	2

8 Conclusion and Future Work

In this paper, we presented a comprehensive set of criteria for evaluating cross-platform development approaches for mobile applications. Results have been compiled in tables, which can be used as references. The ensuing analysis of several cross-platform solutions according to these requirements has shown that PhoneGap is to be preferred, unless the interface necessarily has to resemble native apps as closely as possible.

Table 8. Evaluation of Titanium – Development perspective

D1 **Development environment** Titanium Mobile is tightly integrated into Appcelerator's IDE Titanium Studio [56], which is based on Eclipse. As the IDE is especially tailored to Titanium, it offers auto-completion for the whole Titanium API. Furthermore, it supports deployment to emulators or devices as well as distribution to app stores. Setting up the development environment for Titanium is straightforward but the platform SDKs still have to be installed separately.	3
D2 **GUI Design** GUI design is rather cumbersome and time-consuming, as the user interface is created programmatically via Titanium's JavaScript API. This requires a lot of verbose and repetitive code. Titanium Studio does not offer a WYSIWYG editor to create the interface.	4
D3 **Ease of development** The quality of Titanium's documentation is good. There are numerous, although minimalistic code examples [57]. Nevertheless, initial progress and accustomization to the framework is relatively slow, as a high degree of framework-specific knowledge has to be acquired.	3
D4 **Maintainability** The prototype developed with Titanium has comparatively many lines of code. Anyhow, the app still remains maintainable as Titanium apps can easily be modularized.	3
D5 **Scalability** The aforementioned ability to easily modularize a Titanium app also enables better scalability. Separate files can be included using *Ti.include()* [58] and it is possible to have different windows run in completely separate JavaScript contexts, even though passing data or objects between windows is quite slow.	2
D6 **Opportunities for further development** Source code of apps written for Titanium, at most with the exception of an application's inner logic, can in general not be used with other approaches due to the fact that a large amount of Titanium-specific functions is used. This creates dependencies on the future development of Titanium (compare I4).	5
D7 **Speed and Cost of Development** Developing with Titanium requires a lot of framework-specific knowledge, and does therefore demand a lot of experience. Designing the user-interface is only possible within an emulator or on a device, which slows down development.	5

Mobile Web apps offer a quick and simple entrance into cross-platform development. In summary, the maturity of cross-platform approaches reveals that native development is not necessary when implementing mobile applications. Even if only a single platform is to be supported, a cross-platform approach may prove as the most efficient method due to its low entry barriers.

These low barriers are mainly owed to usage of Web technologies. HTML, CSS, and JavaScript in alignment with Web paradigms are highly suitable for developing cross-platform apps because they are standardized, popular, reasonably simple but powerful and well-supported. Combined with additional measures to utilize the special capabilities of mobile devices, they fulfill the requirements of most mobile scenarios. However, particularly for user interfaces, future research will have to scrutinize the current possibilities. Interfaces of games are an exemplary field where available approaches might fall short.

Table 9. Evaluation of native apps for Android and iOS – Infrastructure perspective

I1 **License and Costs** Android is distributed as open source by the Open Handset Alliance led by Google under a combination of the Apache License 2.0 and GPL [59]. In contrast, iOS is only available in combination with Apple's own hardware and is published under a proprietary end user software license agreement, with some components distributed under GNU GPL and Apple Public Source License. A membership in Apple's developer program for at least USD 99 per year is necessary to be able to deploy apps to end devices or upload them to the app store [60, 61]. Both frameworks can be used to create commercial software.	3
I2 **Supported Platforms** Developing apps natively requires to do so separately for each platform, because programming language and APIs differ. Hence, this approach does not support cross-platform development.	6
I3 **Access to platform-specific features** Direct access to all features.	1
I4 **Long-term feasibility** Studies on the future of the smartphone market forecast that both operating systems will continue to be popular. Developers can rely on large communities, regular bug-fixes and updates.	1
I5 **Look and feel** Full support of the platforms usage philosophy and the employment of native UI elements are self-evident. By definition, everything that can be done with cross-platform approaches is possible natively as well.	1
I6 **Application Speed** The native prototypes are as fast as the prototype developed with PhoneGap. It might be surprising that they are not faster, but this is likely due to heavily optimized implementations of the WebKit library allowing efficient display of Web pages.	1
I7 **Distribution** Native apps can be distributed within the platform-specific app stores, taking into account the provider's – especially Apple's – policies concerning *appropriate* apps.	2

The list of criteria and the subsequent evaluation was based on input from domain experts. This guarantees a high practical relevance of our work. Furthermore, it hints at promising future improvements in cross-platform development approaches for mobile applications. Future research topics include

- keeping track with progress in mobile development frameworks and reassessing existing technologies as the platforms evolve,
- checking whether Web technology can similarly be used for application to different media,
- verifying our results empirically,
- observing how important platform-specific functions might become available through standardized APIs,
- extending and proposing our framework for evaluations in similar contexts, and
- preparing to provide decision advice based on companies' requirements for app developers.

Our future work will specifically address the refinement and evaluation of our approach in close contact with app developers.

Table 10. Evaluation of native apps for Android and iOS – Development perspective

D1 **Development environment**	2
Android apps can be developed with any Java-enabled IDE. Most developers will probably use Eclipse with the corresponding Android plugins [62]. iOS developers require Mac OS and Xcode [63]. Both development environments are mature, although the "ease of installation" is slightly higher when targeting iOS provided there is access to Apple hardware, as no separate installation of an SDK or plugin is required.	
D2 **GUI Design**	1
Both Android and iOS come with a WYSIWYG editor, enabling user interface design without repeatedly having to deploy to an emulator or smartphone. Especially the iOS editor is very mature, concepts like storyboards offer the possibility to visualize and create large parts of the application without having to write a singe line of code.	
D3 **Ease of development**	2
As expected, the documentation of both operating systems is very comprehensive and of high quality. Both provide numerous examples. Getting-started guidelines support beginners, Google regularly publishes blog posts and developers can additionally resort to the very active community. Programmers that already know the underlying programming language can progress rapidly although they need to acquire additional knowledge about the mobile operating system.	
D4 **Maintainability**	3
In terms of LOC, both native prototypes are the most comprehensive. This is due to the very detailed and object-oriented implementation with Java and ObjectiveC in contrast to the concise JavaScript code. As they use object-oriented constructs and separate the code into classes, native apps are (in comparison) easy to maintain, although they might appear to be more heavyweight than their pendants developed in scripting languages.	
D5 **Scalability**	1
In both Android and iOS, program logic and GUI can easily be separated from each other. Furthermore, each view of an app can be developed on its own. This and the object-oriented concept of classes enable development teams to scale even better than with the other frameworks.	
D6 **Opportunities for further development**	6
Code written for one native platform can in general not be ported to another platform due to different programming languages and APIs.	
D7 **Speed and Cost of Development**	5
Developing natively requires the highest degree of specific knowledge and experience. Particularly as an application has to be repeatedly developed for every platform, costs of development are much higher than with cross-platform approaches.	

Acknowledgements. This paper has been written in cooperation with viadee Unternehmensberatung GmbH, Germany. We would like to thank them for continuous support and fruitful exchange regarding the development of mobile applications.

References

1. Gartner: Market share: Mobile communication devices (2012), `http://www.gartner.com/it/page.jsp?id=1924314`
2. Cho, Y.C., Jeon, J.W.: Current software platforms on mobile phone. In: Proc. ICCAS 2007, pp. 1862–1867 (2007)

3. Lin, F., Ye, W.: Operating system battle in the ecosystem of smartphone industry. In: Proc. of the 2009 Int. Symp. on Information Engineering and Electronic Commerce, pp. 617–621. IEEE CS (2009)
4. Tuunainen, V.K., Tuunanen, T., Piispanen, J.: Mobile service platforms: Comparing Nokia OVI and Apple App Store with the IISIn model. In: Proc. ICMB 2011, pp. 74–83. IEEE CS (2011)
5. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. In: Proc. SPSM 2011, pp. 3–14. ACM (2011)
6. Anvaari, M., Jansen, S.: Evaluating architectural openness in mobile software platforms. In: Proc. ECSA 2010, pp. 85–92. ACM (2010)
7. Charland, A., Leroux, B.: Mobile application development: web vs. native. Commun. ACM 54, 49–53 (2011)
8. Goadrich, M.H., Rogers, M.P.: Smart smartphone development: iOS versus Android. In: Proc. SIGCSE 2011, pp. 607–612. ACM, New York (2011)
9. Lakshman, T.K., Thuijs, X.: Enhancing enterprise field productivity via cross platform mobile cloud apps. In: Proc. MCS 2011, pp. 27–32. ACM, New York (2011)
10. David, M.: Flash Mobile: Developing Android and iOS Applications. Focal Press (2011)
11. Anderson, R.S., Gestwicki, P.: Hello, worlds: an introduction to mobile application development for iOS and Android. J. Comput. Sci. Coll. 27, 32–33 (2011)
12. Firtman, M.: Programming the mobile web. O'Reilly (2010)
13. Newman, B.: Are cross-platform mobile app frameworks right for your business? (2011), http://mashable.com/2011/03/21/cross-platform-mobile-frameworks/
14. Behrens, H.: Cross-Platform App Development for iPhone, Android & Co. (2010), http://heikobehrens.net/2010/10/11/cross-platform-app-development-for-iphone-android-co-%E2%80%94-a-comparison-i-presented-at-mobiletechcon-2010/
15. Adobe: Adobe Announces Agreement to Acquire Nitobi (2011), http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html
16. Appcelerator: Appcelerator press release November 1, 2011 (2011), http://www.appcelerator.com/2011/11/appcelerator-raises-15-million-in-funding/
17. jQuery Mobile (2011), http://jquerymobile.com/
18. Sencha Touch (2011), http://www.sencha.com/products/touch/
19. PhoneGap (2011), http://www.phonegap.com/
20. Apache Cordova (2012), http://incubator.apache.org/cordova/
21. PhoneGap, Cordova, and what's in a name? (2012), http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/
22. About PhoneGap (2011), http://phonegap.com/about
23. Appcelerator Titanium Platform (2012), http://www.appcelerator.com/platform
24. Titanium Mobile open source project (2012), https://github.com/appcelerator/titanium_mobile
25. Rhodes (2012), http://www.motorola.com/Business/US-EN/RhoMobile+Suite/Rhodes
26. iPhonical (2010), http://code.google.com/p/iphonical/
27. applause (2012), https://github.com/applause/
28. XMLVM (2012), http://www.xmlvm.org/android/

29. 15 most important considerations when choosing a web development framework (2009), http://net.tutsplus.com/tutorials/other/15-/
30. Pfeiffer, D.: Which cross-platform framework is right for me? (2011), http://floatlearning.com/2011/07/which-cross-platform-framework-is-right-for-me/
31. Lukasavage, T.: Adobe & PhoneGap: Makes sense, mostly (2011), http://savagelook.com/blog/portfolio/adobe-phonegap-makes-sense-mostly
32. Kassinen, O., Harjula, E., Koskela, T., Ylianttila, M.: Guidelines for the implementation of cross-platform mobile middleware. International Journal of Software Engineering and Its Applications 4 (2010)
33. Sencha ext JS (2012), http://www.sencha.com/store/extjs/
34. Koch, P.P.: There is no WebKit on mobile (2009), http://quirksmode.org/blog/archives/2009/10/there_is_no_web.html
35. Pilgrim, M.: Dive into HTML5: Local storage (2011), http://diveintohtml5.info/storage.html
36. CSS theme for iPhone (2011), http://www.predic8.com/iphone-css-layout-theme.html
37. W3C: HTML5: offline web applications (2012), http://www.w3.org/TR/html5/offline.html
38. Apple: App Store review guidelines (2012), https://developer.apple.com/appstore/guidelines.html
39. Firebug (2012), http://getfirebug.com/
40. Dornbierer, C., Ong, J., Boon, P.: Cross-platform mobile application development (2011), http://www.adnovum.ch/pdf/slides/adnovum_jazoon2011_mobile_engineering.pdf
41. Murphey, R.: On jQuery & large applications (2010), http://rmurphey.com/blog/2010/08/09/on-jquery-large-applications/
42. PhoneGap license (2012), http://phonegap.com/about/license/
43. jQuery project license (2012), http://jquery.org/license/
44. PhoneGap support (2012), http://phonegap.com/support#support-packages
45. jQuery Mobile graded browser support (2012), http://jquerymobile.com/gbs/
46. PhoneGap: Supported features (2012), http://phonegap.com/about/features/
47. PhoneGap: FAQ (2012), http://phonegap.com/faq
48. PhoneGap: Build (2012), https://build.phonegap.com
49. PhoneGap: API reference (2012), http://docs.phonegap.com/en/1.8.0/index.html
50. jQuery Mobile documentation (2012), http://jquerymobile.com/demos/1.1.0/
51. Rolling releases: How Apache Cordova becomes PhoneGap and why (2012), http://phonegap.com/2012/04/12/rolling-releases-how-apache-cordova-becomes-phonegap-and-why/
52. Titanium: Plans & pricing (2012), http://www.appcelerator.com/plans-pricing
53. Titanium Mobile 2.0.1.GA release notes (2012), http://docs.appcelerator.com/titanium/release-notes/?version=2.0.1.GA

54. Titanium API (2012), http://docs.appcelerator.com/titanium/2.0/index.html#!/api
55. Whinnery, K.: Comparing Titanium and PhoneGap (2012), http://developer.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap.html
56. Titanium Studio (2012), http://www.appcelerator.com/platform/titanium-studio
57. Titanium documentation (2012), http://docs.appcelerator.com/titanium/2.0/index.html
58. Titanium include API (2012), http://docs.appcelerator.com/titanium/2.0/index.html#!/api/Titanium
59. Google: Android open source project (2012), http://source.android.com/
60. Apple: iOS developer program (2012), http://developer.apple.com/programs/ios/
61. Chudnov, D.: A mobile strategy web developers will love. Computers in Libraries 30, 24–26 (2010)
62. Android Development Tools plugin for Eclipse (2012), http://developer.android.com/sdk/eclipse-adt.html
63. Xcode 4 (2012), https://developer.apple.com/xcode/index.php
64. Fishburn, P.C.: Additive utilities with incomplete product sets: Application to priorities and assignments. Operations Research 15, 537–542 (1967)