Evaluating Frameworks for Creating Mobile Web Apps

Henning Heitkötter, Tim A. Majchrzak, Benjamin Ruland and Till Weber
Department of Information Systems, University of Münster, Münster, Germany

Keywords: App, Mobile Web App, Framework, Cross-platform, Comparison, Evaluation, User Interface.

Abstract: Mobile Web apps are an alternative to native apps when developing mobile applications for more than one platform. They can be accessed from almost any current smartphone and tablet using a browser. However, developing a Web app adapted to the particularities of mobile devices such as limited screen size and touch-based interaction requires consideration and effort. Several frameworks with different strengths and weaknesses support creating mobile Web apps. In this paper, we develop a set of criteria to be met by mobile Web frameworks of high quality. Based on these criteria, we evaluate jQuery Mobile, Sencha Touch, The-M-Project, and Google Web Toolkit combined with mgwt, thereby assessing their suitability for certain situations. We find Sencha Touch suited for mobile Web apps of moderate and increased complexity, while jQuery Mobile is a good starting point for apps primarily concerned with a mobile user interface.

1 INTRODUCTION

Mobile devices such as smartphones and tablets are still gaining popularity among consumers and enterprises (Gartner, 2012). In some areas they are already replacing – rather than just complementing – PCs and laptops. For software developers, mobile devices are a blessing and a curse at the same time. Obviously, there is a demand for new *apps*, and the devices offer novel possibilities such as GPS positioning and – of course – truly mobile usage. At the same time, a variety of platforms such as Android or iOS (Lin and Ye, 2009), a variety of frameworks, and a lack of best practices make it cumbersome to implement them. If multiple platforms have to be supported, effort increases almost linearly with the number of platforms. This leads to a need for *cross-platform* development.

In the last decade, companies have significantly invested in Web technology (Chatterjee et al., 2002; Basu et al., 2000). More and more, applications that do not require rich clients for profound reasons (e. g., integrated development environments or professional media editing tools) are provided as Web apps. Users merely require a Web browser to use them.

The ambition to support multiple platforms and existing knowledge of Web technologies make Web apps an interesting choice for app development. In general, they promise to offer almost as good platform support as *native* apps. The latter are developed using a platform's source development kit (SDK) and should be employed if truly native performance and look & feel are desirable. Unfortunately, the requirements for Web apps on mobile devices differ from Web apps in general. To effectively implement them or to make existing Web apps suitable for mobile devices, sophisticated framework support is advisable. An abundance of available frameworks hampers an easy selection, though. Moreover, there is hardly any guidance for informed decision-making.

To fill this gap, it is important to evaluate frameworks based on sound criteria. In this paper, we describe our criteria-based evaluation process, which can be sketched as follows: Based on typical requirements of apps and on an extensive set of information resources, we derive eleven qualitative criteria. These criteria, accompanied by corresponding assessment guidelines, are then used to evaluate four frameworks. Background information and, more importantly, own experience are the foundation for judging to what degree a framework fulfills a certain criterion.

Our paper makes a number of contributions. First, we describe a novel set of evaluation criteria useful beyond their application in this paper. Second, we evaluate the frameworks in detail and give operationalizable decision advice. Third, we contribute to the theory with a discussion of implications of our research. By outlining which approaches look promising and where current frameworks lack features, we highlight areas for further research and development.

Heitkötter H., Majchrzak T., Ruland B. and Weber T..
Evaluating Frameworks for Creating Mobile Web Apps.
DOI: 10.5220/0004356702090221
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 209-221
ISBN: 978-989-8565-54-9

This paper is structured as follows. Related work is studied in Section 2. Section 3 introduces the concept of mobile Web apps and presents the frameworks analyzed later, including their general characteristics. Our evaluation criteria are developed in Section 4 and then used in Section 5 to evaluate the frameworks. Section 6 discusses and summarizes the findings, before we conclude in Section 7.

2 RELATED WORK

Our kind of research is literature-driven. Therefore, relevant citations are provided in the corresponding paragraphs throughout this paper. In this section, we distinguish our work from existing approaches. The main observation is that there are no all-encompassing reviews based on scientific criteria. Rather, many papers evaluate single frameworks in isolation or a combination thereof. This most likely is owed to the novelty of the field of research. Nevertheless, these papers have made valuable contributions and in parts have been influential for our work.

Several papers evaluate technologies for Web apps such as HTML5 (HTML5, 2012). With additions like offline support, HTML5 is particularly suited for mobile Web apps. This is also reflected in recently published textbooks (e. g., (Oehlman and Blanc, 2011)). Assessment of HTML5 is positive (Harjono et al., 2010; Melamed and Clayton, 2009). Obviously, it is mature enough for widespread usage (Lubbers et al., 2011). Typically, HTML5 and JavaScript are utilized together (Meyer, 2011).

Frameworks are often evaluated in the context of app development. For example, in a comparison of Web apps and native apps it makes sense to mention jQuery mobile (Charland and Leroux, 2011). This does not help to compare jQuery mobile to competing approaches, though. The same applies to work on HTML5 that includes framework recommendations (e. g., (Curran et al., 2012)). The paper by Smutny goes a step further and briefly compares several frameworks (Smutny, 2012). However, he does not propose a catalogue of criteria for doing so.

Another thread of research is scenario-based evaluation of existing technologies. For example, Zibula and Majchrzak (2012) use HTML5, jQuery Mobile, and PhoneGap to build an app for smart metering. However, while such papers help to understand the feasibility of possible technology stacks, they do not provide a greater image such as our work.

Our previous paper on evaluating cross-platform development approaches in general (Heitkötter et al., 2012) presents complementary work. It thereby helps to make a decision on a higher-level: Web app *or not*. The research design is similar to this article, while the outcome is of course different. Ohrt and Turau (2012) present a broad comparison of cross-platform development tools, but do not consider mobile Web apps.

3 MOBILE WEB APPS

This section examines mobile Web apps as a cross-platform approach to app development and introduces four frameworks that will be analyzed later.

3.1 General

A Web application, short *Web app*, is a Web site that provides an application within browsers, as opposed to static content (Connors and Sullivan, 2010, Sect. 1.3.2). It is built with Web technologies such as HTML5, CSS, and JavaScript to enable a dynamic experience. A *mobile* Web app is intended to be used on mobile devices. It may be a mobile-optimized version of an existing Web app. In contrast to standard mobile apps, mobile Web apps are not installed on the device (e. g., via an *app store*) but accessed through the browser. Although built with the same set of technologies, special requirements due to the mobile environment call for a different approach and specific optimizations.

Mobile-specific requirements mainly stem from limited screen size available on mobile devices, a different style of user interaction through touch gestures, and the mobile context. The smaller screen of smartphones and, to a lesser extent, of tablets requires a different user interface (UI) layout and mobile versions of typical HTML elements. For example, a multi-column layout is usually not feasible in mobile Web apps; instead, fixed toolbars for header or footer can provide universal navigation. Users interact with their devices primarily through touch interaction, which replaces the traditional pointer-based interaction combined with keyboard input. This requires several adaptations. UI elements have to be of sufficient size if users shall select them. Mobile Web apps should not expect the user to enter a large amount of text. They should, however, support gestures as an additional means of interaction. The mobile context includes more particularities that have to be accounted for such as limited hardware resources and instable or slow network connections. Hence, mobile Web apps should *optimize* network requests.

Combined with nearly 100 best practices recommended by the World Wide Web Consortium (W3C)

for developing mobile Web pages (Connors and Sullivan, 2010; Rabin and McCathieNevile, 2008), the aforementioned requirements highlight the need for Web frameworks that support the development of mobile Web apps. Hence, there are various suchlike frameworks. In order to select promising frameworks for evaluation, we studied Web sites and literature dealing with mobile Web frameworks. *jQuery Mobile* and *Sencha Touch* were mentioned most often and will be analyzed in the following. Third-placed *jQTouch* bears close resemblance to jQuery Mobile and is not investigated further, because its evaluation would not provide additional insight. Several frameworks followed with significantly less mentions, of which we selected *The-M-Project* as a promising alternative. *Google Web Toolkit* combined with *mgwt* completes the set of candidates. As apps using GWT are programmed in Java, this combination represents a differing approach to mobile Web development.

As they run within a browser environment, mobile Web apps have some limitations, mostly regarding access to device-specific features and hardware. Depending on requirements, they are *not always* the optimal choice for cross-platform app development, as demonstrated by Heitkötter et al. (2012).. Several popular cross-platform development frameworks are based on Web technologies. They follow a so-called hybrid approach and package mobile Web apps with a runtime that gives access to device features. Such development projects often utilize mobile Web frameworks as well. Hence, we analyzed in how far frameworks can be combined with PhoneGap (2013), also named Apache Cordova, a popular hybrid framework.

3.2 jQuery Mobile

jQuery Mobile (2013) makes the user interface of Web pages accessible on mobile devices. To develop an app with jQuery Mobile, developers merely need to use HTML5; at first glance, development is quite similar to Web development in general. By assigning specific attributes and values to HTML elements, they can be transformed into mobile-optimized UI elements or get a certain role, such as *header* or *button*. Before the app is rendered in the client's browser, jQuery Mobile enhances the HTML markup based on values of these attributes by adding additional markup and CSS classes. Besides UI components, jQuery Mobile provides animations and JavaScript support for touch events. It does not include an API for advanced features such as data binding or internationalization. It does, however, incorporate the popular JavaScript library jQuery (2012).

jQuery Mobile aims to deliver a uniform user interface and high-class JavaScript to a wide range of mobile devices. All prevalent operating systems, namely Android, iOS, BlackBerry OS, Windows Phone and Symbian, are supported. Mobile Web apps using jQuery Mobile can also be packaged with PhoneGap. The framework uses *progressive enhancement* (Firtman, 2012), which adjusts the presentation according to supported features of displaying devices. It thus enables jQuery Mobile apps to run on nearly every smartphone browser.

Created in November 2010 by the jQuery Project (2012), jQuery Mobile is since maintained as open source under MIT license. Beneath the jQuery Project, it is supported by companies like Adobe, Mozilla Corporation, Palm, BlackBerry, and Nokia. It is part of a large ecosystem, which, besides others, includes a *ThemeRoller* for developing custom styles. Our review assesses version 1.2.

3.3 The-M-Project

The-M-Project (2013) provides a JavaScript framework for creating mobile Web apps with a Model-View-Controller (MVC) architecture. Apps are written entirely in JavaScript, without HTML or CSS. Not only data model and controller logic are implemented in JavaScript, but also the user interface. The JavaScript code of an application builds up the HTML at runtime in the client's browser, resorting to jQuery Mobile for creating the UI. In addition to means for programmatic UI definition, The-M-Project's API provides features such as data binding, event handling, AJAX requests, and internationalization.

It is licensed under MIT License and primarily developed by Panacoda. The first version was released in 2011. Our evaluation examines version 1.2. Since mobile Web apps using The-M-Project only require HTML5, they are available for all platforms where jQuery Mobile is supported and can also be packaged with PhoneGap ("The-M-Docs. Native Packaging", 2012). The-M-Project includes Espresso, a build tool simplifying the development lifecycle. It sets up the initial project structure and creates required components. Furthermore, Espresso supports optimizing, packaging, and deploying the mobile Web app.

3.4 Sencha Touch

Sencha Touch (2012) enables the structured development of mobile Web apps by only using JavaScript, similar to The-M-Project. Main elements of the framework are *components*: an extensive inheritance hierarchy of components contains all functionality. Hence, developing with Sencha Touch mostly con-

sists of instantiating and configuring these components. Since most components allow nesting other components or docking them to their sides, even complex structures can be realized. This procedure applies to creating the user interface, where layouts determine how nested components are aligned, and to dynamic parts. A MVC architecture supports modularity and the utilization of dynamic data.

Sencha Touch was announced in June 2010 and is developed by Sencha Inc. We examine Version 2, released in May 2012. It is compatible with Android, iOS, and BlackBerry, but provides no explicit support for Windows Phone or Symbian. Packaging with PhoneGap is possible (Dougan, 2012). Sencha Cmd is a build tool for projects using Sencha Touch. It can be used to set up, build, and deploy a project.

3.5 Google Web Toolkit with mgwt

mgwt (2013) adds support for mobileWeb apps to the popular Google Web Toolkit (2012). GWT allows to develop complex Web apps in Java, which GWT then compiles to JavaScript that can be run in most modern browsers. GWT's Java API abstracts from browser differences and low-level AJAX or DOM operations, aiming to increase productivity. In addition to the extensive features of the Java programming language, it offers support for AJAX requests, management of browser history, and internationalization. Several libraries extend GWT with additional widgets and functionality. Since GWT lacks UI elements optimized for mobile devices, mgwt extends GWT with corresponding features, placing a strong focus on native look & feel and performance. To this end, mgwt provides developers with an additional, mobile-specific API for building up the user interface programmatically in Java, including animations.

GWT was released in 2006 and is developed by Google. Developed since 2011, mgwt mainly relies on a single developer. Both are licensed under Apache License 2.0. Versions examined are GWT 2.5.0 and mgwt 1.1.1. GWT supports most modern desktop browsers (GWT FAQ, 2012). mgwt focuses on mobile browsers based on WebKit, which are available for iOS, Android, and Blackberry. Native themes are provided for iOS and Android. Mobile Web apps can be packaged as PhoneGap applications with GWT-Phonegap (2013). For GWT, there are numerous tools helping developers in all phases of the development lifecycle, including plugins for the popular Eclipse IDE, debugging facilities, and a graphical UI builder. Setting up a project that uses mgwt is slightly more complicated. No UI builders for mgwt are available.

4 CRITERIA

Selecting a Web framework from a set of alternatives constitutes an optimization problem. A decision-maker has to select from a limited choice of alternatives, so that his utility or that of his company is maximized. To deal with the complexity of measuring utility directly, most decision-making methods split up the utility-wise consequences of a selection into the consequences of disjoint criteria on utility (Triantaphyllou and Mann, 1989). The utility stemming from individual criteria can eventually be combined using an additive function (Fishburn, 1967) or complex functions such as the *Analytic Hierarchy Process* (Saaty, 1986). In any case, identifying separate criteria simplifies evaluation because each criterion can be examined on its own. Moreover, it improves decision-making because the decision process is dissected into manageable components. Decision-makers can weight criteria according to their needs. Combining all requirements of mobile Web frameworks into a single measure would prove rather difficult and opaque. Hence, the first step of our evaluation consisted of developing a set of criteria.

In order to arrive at meaningful criteria, a goal hierarchy guided our criteria elicitation. The overall goal was to allow decision-makers to select the optimal mobile Web framework depending on differing requirements. Two mostly separate sub-goals further define the quality of such a framework: first, functionality and usability of mobile apps created with the framework and second, the developer experience when developing mobile Web apps using the framework. The first focuses on the users' perspective and their experience using an app, which literature considers an important factor of acceptance (Tarasewich, 2003; Gong and Tarasewich, 2004). The user perspective is of course highly relevant for the developer as well, but only mediated through the requirement to build a user-friendly app. In contrast, the latter goal takes into account the developers' perspective and other decision-relevant factors that foremost affect developers and the development costs of an app project. Based on these two goals we grouped the criteria into *user* and *developer* perspective.

Our process of deriving a set of criteria suitable for evaluating mobile Web frameworks included several sources of information. We set out with a review of existing articles and Web sites that give decision advice for choosing frameworks related to Web *or* mobile development in general (Heitkötter et al., 2012; Olaru, 2007; Walsh, 2008; Lennon, 2010). Since those areas have some overlap with our topic, the comparisons provided some insights into typical re-

quirements of frameworks, which influenced our criteria. Special consideration was given to particularities and challenges due to the mobile environment and to Web-specific requirements, both already outlined in Section 3.1. Our experience developing apps further contributed to the set of criteria, as well as expectation formulated by partners from industry in interviews, ongoing discussions, and joint projects.

This process resulted in two kinds of criteria: *binary* and *qualitative*. Binary criteria are concerned with questions whether a framework possesses a required property and can be answered with yes or no. These include, for example, if a framework supports a certain mobile platform or its compatibility with PhoneGap. They have mostly been answered during the introduction of the frameworks in Section 3. Binary criteria restrict the set of suitable frameworks prior to an evaluation. Qualitative criteria deal with the quality of a framework with respect to its fulfillment of requirements. Typically, quality is graded on an ordinal scale. Qualitative criteria are less obvious and can typically not be inferred reliably from descriptions of a framework. Instead, evaluating a framework with respect to these criteria requires intensive engagement with a framework. At the same time, they are highly decision-relevant. Hence, this paper focuses on qualitative criteria.

Tables 1 and 2 display our resulting set of qualitative criteria, divided into developer and user perspective. Each criterion has a name and is identified through a letter for the perspective and a running digit. A short description motivates each criterion and lists indicators, i. e., factors influencing its evaluation.

5 EVALUATION

In this section, we present the results of our evaluation. We assessed the four mobile Web frameworks according to the criteria outlined in Section 4. Results are described separately for each framework in the following subsections. Beforehand, the next subsection outlines our evaluation process. Table 3 gives an overview of all individual assessments in terms of grades, organized along frameworks and criteria.

5.1 Evaluation Process

Our evaluation of each framework consisted of two steps: collecting information about the framework in question and using it to develop prototypical apps. Publicly available information such as documentation, community resources, and reviews was helpful in gaining a first impression of the quality of a framework. Certain criteria, for example license and costs or long-term feasibility, can even solely or best be assessed through such information sources. For other criteria, the information represented a good starting point for the ensuing in-depth scrutiny through development experience. For example, typical problems as observable via community resources of a framework or feedback from our industry partners hinted at potential benefits and drawbacks. We developed mobile Web apps for managing contacts and schedules. These apps were intentionally prototypical. Their set of requirements had been compiled so that profound insight was gained while implementing them. Requirements included multiple screens with a native look & feel, advanced widgets and layout, local storage, and asynchronous loading.

Based on these experiences, two reviewers jointly assigned grades on a scale from 1, *very good*, to 6, *very poor* for each criterion. They complement the textual evaluation for a quick overview. As highlighted in Section 4, each criterion is associated with a set of requirements and guidelines, which formed the basis of the assessment of each framework. This process ensured that evaluation was as objective as possible given the qualitative nature of the task at hand.

5.2 jQuery Mobile

jQuery Mobile uses the MIT License, which supports both open-source and closed-source projects. As no other costs accrue for support or development tools, *License and Costs* are suitable for any project (grade 1). In terms of *Long-Term Feasibility*, jQuery Mobile meets the demand of high popularity as it is frequently mentioned in reviews, literature, and general developer forums. jQuery Mobile cites several notable references such as Ikea or Disney World. As evident from the success of jQuery and thanks to several supporting firms, the development team promises a stable and steady further development. Furthermore, short update cycles in recent times predict a positive update behavior in future, so that, overall, jQuery Mobile should remain viable long-term (1).

The documentation covers all available features in a concise but understandable way. While it does not showcase sample applications or tutorials, several textbooks, articles, and tutorials by third-party authors are referenced. Support is available from the highly frequented jQuery Mobile Forum (2013) with about 300 topics per month and from external support forums, in which jQuery Mobile seems to be a relevant topic (“Stack Overflow. Tag jQuery Mobile”, 2013). Thus, the criterion *Documentation and Sup-*

Table 1: Criteria of the developer's perspective.

D1	**License and Costs** Costs for obtaining a framework and employing it in commercial apps influence whether a framework is suitable for a certain app or a particular company. Hence, this criterion examines licensing costs that accrue for developing and publishing a commercial app based on the respective framework. Additionally, costs of support inquiries are considered (actual development costs are assessed by criterion *D5*). The optimal case would be an open-source framework under a permissive license such as MIT License (1988) or Apache License (2004). Copyleft licenses (Sen et al., 2011) such as GNU General Public License (2007) might complicate commercial, closed-source projects. This criterion is especially relevant for smaller projects and companies; it might also be relevant from a development perspective (Sen et al., 2008).
D2	**Long-term Feasibility** The decision for a framework represents a significant investment because specific know-how needs to be acquired and source code of apps will be tied to the framework. Hence, developers will prefer a framework that will most likely be available in the long term. A framework needs continuous updates, especially in view of rapidly changing browsers and Web technologies. Indicators of long-term feasibility are popularity, update behavior, and the development team. Popularity can be assessed through a high diffusion rate among app developers and recognition in the developer community, for example through reviews. A positive update behavior is marked by short update cycles and regular bug-fixes. A framework with a strong development team, ideally backed by several commercial supporters, is more likely to continue to exist in the future.
D3	**Documentation and Support** Documentation and further support channels assist developers in learning and mastering a framework. Assistance is not only required when starting to use a framework, but also to efficiently employ its API and advanced concepts. Therefore, a documentation of good quality provides tutorials and a comprehensive, well-structured reference. For popular frameworks, textbooks might provide a good starting point. Besides, other means of support such as community-driven forums or paid assistance help in case of special problems.
D4	**Learning Success** Time and effort needed to comprehend a framework directly affect its suitability. While a good documentation (*D3*) may enhance learning success, learning inherently depends on the inner characteristics of a framework, i. e., its accessibility and comprehensibility. Hence, the learning success is examined separately. It mainly depends on the subjective progress of a developer during initial activities with a framework. Intuitive concepts, possibly bearing resemblance to already known paradigms, can be mastered quickly. To a minor extent, this criterion also considers the effort needed for learning new concepts after initial orientation.
D5	**Development Effort** The cost for developing apps mostly depends on the development effort needed, assuming a basic familiarity with the framework. While certain development phases such as requirements elicitation or design are largely independent of the framework used, it directly influences the implementation. Hence, the development effort is characterized by the time needed for implementing apps with the framework. Indicators for a framework that ease development are expressive power, an easy-to-understand syntax, reusability of code, and good tool support. The latter includes an Integrated Development Environment (IDE), which facilitates implementation and possibly GUI design, as well as debugging facilities.
D6	**Extensibility** In view of both evolving requirements and a changing environment, it may be necessary to extend a framework with additional functionality, either during initial implementation or in later iterations. This will be easier and more stable if a framework offers corresponding features such as a plug-in mechanism. As a last resort, app developers might adapt the source code of the framework itself, provided it is available. Besides considering the existence of extensibility measures, this criterion assesses their usefulness and accessibility.
D7	**Maintainability** Web apps can and will be updated regularly. Therefore, their implementation must be maintainable over a longer period. This criterion is positively correlated with comprehensibility of the source code and its modularity. Both indicators depend on the framework used to implement the app. A framework that allows for concise but understandable code will improve comprehensibility. Modularity requires the possibility to separate different parts of an app into distinct units of code.

port is assessed as good (2).

Looking at the *Learning Success*, one can easily implement a simple application with jQuery Mobile, as only standard HTML and few custom, intuitive attributes are needed. For developing a richer dynamic app, skills in JavaScript and jQuery are required in addition. As Web developers are likely to have such previous knowledge, a quick learning success can be

Table 2: Criteria of the user's perspective.

U1 **User Interface Elements** From an app user's perspective, elements of the UI should be well-designed and optimized for mobile usage. Hence, a mobile Web app framework needs to provide high-quality elements for important tasks. On the one hand, this criterion assesses whether a framework offers mobile versions of common structural elements, i. e., widgets such as buttons or text fields and their layout in containers, as well as their quality. Structural elements need to address limited screen sizes and particularities of touch-based interaction. On the other hand, a framework should support behavioral UI elements such as animations and gestures.
U2 **Native Look & Feel** User acceptance of a Web app, also compared to a native app, often depends on a native look & feel. In contrast to a typical Web site, apps with a native UI have a platform-specific appearance and behavior. As this is an often mentioned requirement of apps, this criterion assesses whether a framework offers support for a native look & feel. Optimally, a framework would provide different, platform-specific themes, at least for Android and iOS. If that is the case, we examine how closely these resemble truly native UIs. Otherwise, the framework should provide means to efficiently style its UI elements and implement themes.
U3 **Load Time** The time required to load a Web app is important to users in view of slow and instable network connections on mobile devices. In contrast to native apps, Web apps moreover are not installed but retrieved upon access. Load times partly depend on the code size of the framework and on the typical verbosity of code using the framework. A framework might provide means to reduce initial load time such as support for asynchronous requests (AJAX) or storing application and user data locally on the device (via features of HTML5).
U4 **Runtime Performance** The performance at runtime (after loading) informs the overall impression of an app. Since they run in a *sandbox*, Web apps might suffer from a comparatively low performance. Hence, the overall performance of a framework, as subjectively experienced by users, is important. The UI elements need to react quickly to user interactions and animations should be smooth for a high-quality user experience.

Table 3: Assessment summary.

Symbol	Criterion	jQuery Mobile	The-M-Project	Sencha Touch	GWT + mgwt
D1	License and Costs	1	1	2	1
D2	Long-term Feasibility	1	3	2	4
D3	Documentation and Support	2	2	1	3
D4	Learning Success	1	3	3	3
D5	Development Effort	4	3	2	2
D6	Extensibility	3	5	1	2
D7	Maintainability	4	2	1	2
U1	User Interface Elements	2	1	1	2
U2	Native Look & Feel	5	6	4	1
U3	Load Time	2	3	3	1
U4	Runtime Performance	3	3	2	1

achieved with minor effort. Further education mostly deals with jQuery core elements and might therefore also be achievable with little effort. All in all, learning and mastering jQuery Mobile is easy (1).

Developing static applications solely in HTML required little time and effort. A simple syntax and easy debugging also speeds up development. Dynamic applications, however, require pages to be altered prior to rendering, typically by the use of DOM manipulation. While this is no obstacle to small applications, the *Development Effort* might increase for bigger projects. There are no special IDEs that support development with jQuery Mobile. jQuery Mobile does not provide APIs for advanced functionality such as data binding or internationalization and focuses on creating mobile user interfaces, so that it helps little when developing complex Web apps (4).

jQuery Mobile can be extended via plug-ins that either add custom methods to jQuery Mobile's JavaScript API or provide additional widgets. jQuery Mobile reuses the plug-in mechanism of jQuery but does not state its corresponding API comprehensively. Only a medium number of extension is referenced on the project's homepage. Hence, the framework's *Extensibility* is satisfactory (3). The source code of a Web app built with jQuery Mobile mainly consists of

HTML code, which tends to be comprehensive. As other parts of the source code are mainly written in JavaScript and no structural concepts are provided, the readability of these parts depends on the programming style of the developer. In terms of modularity, jQuery Mobile lacks support to separate different parts of an app into individual sections, so that, overall, *Maintainability* is limited (4).

jQuery Mobile provides various *User Interface Elements* such as form elements, lists, tool bars, and grid layouts. Their quality is sufficient as they are adapted to small screen sizes and touch-based interaction, but does not reach the quality of native UI elements. Page transitions can be enhanced by a set of animations and touch gestures can be detected by specific JavaScript-events. All in all, the support for mobile UIs is good (2). The creators of jQuery Mobile aim for a unified user interface. Therefore, the look & feel purposefully differs from the native appearances of each platform and no platform-specific templates are provided. A custom adaption of native designs is possible, since the design is mainly influenced by CSS, but tends to be rather complex, even when assisted by the ThemeRoller. Hence, the criterion *Native Look & Feel* is not well-fulfilled (5).

The *Load Time* of a jQuery Mobile application as experienced by users is rather short. On the one hand, this effect is due to the small size of the framework, which is less than 100 KiB in its minified form. On the other hand, jQuery Mobile reduces the processing time of browsers, so that the Web app is displayed faster. Several techniques like AJAX or prefetching help to reduce interruptions during usage, while keeping the initial load time down. Storing the whole application data on a mobile device is possible but not assisted. In summary, jQuery Mobile applications load comparatively quickly (2). On high end devices such as Apple's iPhone 4S or Samsung's Galaxy S2, experienced runtime performance can hardly be distinguished from native apps, since animations run fluid and response to user interaction is almost immediate. Mid-end devices such as Samsung's Galaxy Ace, show, however, performance issues, as scrolling occurs bumpy and animations are barely visible. On average, *Runtime Performance* is satisfactory (3).

5.3 The-M-Project

License and Costs of The-M-Project are favorable for all kinds of projects (1). *Long-term Feasibility* heavily depends on the major backer of the project, Panacoda. New versions appeared in regular intervals over the last year ("The-M-Project. github repository", 2013). The-M-Project's popularity is difficult to assess as the homepage presents no references. Its forum ("The-M.Project. Google Groups", 2013) shows steady, moderate interest (approximately 20 topics per month over the last half year), its repository is watched by more than 500 developers. All in all, The-M-Project's long-term view is solid (3).

The-M-Project's documentation provides tutorials as well as a reference of concepts and of the API. Overall, the documentation is well-structured and extensive; at times, in-depth information is missing, e. g., on the class *Model*. Several sample apps accompany the documentation. Additional support is available through the mentioned community forum, which usually provides timely answers. In summary, *Documentation and Support* are good (2). When familiarizing themselves with the framework, Web developers need to get used to programming only in JavaScript without HTML or CSS, and, hence, also learn The-M-Project's API. The build tool Espresso helps with getting started, as does the documentation. Mastering the concepts needs detailed information, which the documentation does not provide for all parts. The overall *Learning Success* achieved with The-M-Project is slowed due to the extensive API knowledge required and the unusual approach to Web development (3).

The-M-Project advocates many structural requirements on Web app projects, partly due to the MVC architecture. For smaller projects, this overhead might complicate development more than necessary, while The-M-Project's advanced concepts help when implementing complex apps. In general, a large amount of boilerplate code is required, which partially can be generated by Espresso. Defining the UI programmatically in JavaScript is quite verbose and cumbersome. On the other hand, advanced features of The-M-Project, such as content binding, free developers from implementing complex logic manually. JavaScript in general and The-M-Project in particular lack sophisticated development environments that offer code completion and other helpful features. Since JavaScript gives developers great flexibility, a larger code base requires strict development guidelines to maintain structure. On average, *Development Effort* with The-M-Project is slightly increased (3), mainly due to the programmatic UI creation.

The-M-Project offers no means for extending the framework. The only way to adapt the framework is modifying the openly available source code. Hence, *Extensibility* is poor (5). Thanks to the well-defined architecture prescribed by The-M-Project, apps can be maintained quite well. They typically have a modular design and separated concerns. The rather verbose source code might decrease comprehensibility, but overall, *Maintainability* is good (2).

As The-M-Project reuses jQuery Mobile for building up the UI, most of the assessment of jQuery Mobile with respect to widgets, animations, and gestures applies here as well. The-M-Project provides additional widgets not available in jQuery Mobile such as a date picker or a map view. Hence, the set of *User Interface Elements* is even better than jQuery Mobile's (1). The-M-Project uses the default theme of jQuery Mobile and allows optional adaptation with CSS. Hence, a *Native Look & Feel* is not supported, either, and changing the style is less comfortable (6).

The complete The-M-Project framework is rather large with a size of close to 400 KiB (minified JavaScript, CSS, and images) because it bundles several libraries with its own source code. Apps built with The-M-Project tend to be comparatively large, in part owed to the boilerplate code required for model, view, and controller. At the same time, The-M-Project provides good support for minifying and making apps available offline via HTML5's *application cache*, as Espresso generates required manifests automatically. This simplifies reducing the *Load Time* to acceptable levels (3). The *Runtime Performance* of apps is satisfactory and similar to jQuery Mobile (3); no obvious stuttering or lag is discernible.

5.4 Sencha Touch

Sencha Touch is licensed under either GPL or a free-of-charge commercial license that allows closed-source distribution. Enhanced support and a specialized development tool can additionally be purchased at extra charge, so that the criterion *License and Cost* is well-fulfilled (2), although Sencha Touch is less open than the other frameworks. Long-term feasibility of Sencha Touch relies almost exclusively on Sencha Inc. Since the framework is a major product of Sencha and the company is not only dependent on supporters but also on own revenues, a suspension is unlikely. Sencha Touch has an equally high popularity as jQuery Mobile. Frequent and major updates in the past year point to a short update cycle. Summing up, *Long-term Feasibility* seems stable, but is heavily dependent on a single, albeit established company (2).

The documentation extensively covers the API, with helpful, modifiable examples and good structure. In addition, one can access several tutorials and sample apps. A support forum shows high activity (in the last year about 700 topics created per month). Besides, Sencha Touch offers a special support forum and telephone assistance with costs, so that the criterion *Documentation and Support* is covered well (1). When learning Sencha Touch, one has to deal with JavaScript and the framework's API. Developers need to familiarize themselves with the component structure and the way in which components are instantiated and linked to each other. As this differs from common Web development approaches, initial learning progress is quite slow. After being familiar with the API, further learning is, however, easier, due to good documentation and well-structured components. Overall, *Learning Success* is satisfactory (3).

Sencha Touch requires a lot of effort for developing small applications due to structural overhead by concepts like MVC. In contrast, larger applications benefit from these concepts. Both MVC and the possibility to nest components simplify structuring an app. Additionally, an IDE is available with costs (Sencha Architect, 2013). It provides a WYSIWYG UI editor, a code editor, and supplementing tools. Summarizing, the average *Development Effort* is low, mainly due to the structured approach (2).

In terms of *Extensibility*, developers can add custom components that inherit from other components. Besides, Sencha Touch provides an interface for adding plug-ins to components that alter their functionality. Hence, the framework's extensibility is very good (1). The expressive API, as evident for example in self-explanatory parameters, leads to comprehensible source code. As separation of concerns and the allocation into various files facilitate modularity, *Maintainability* is, overall, very good (1).

The set of UI elements such as widgets, animations, or gestures is wide and of high quality. It can be compared to that of The-M-Project. As nesting of elements is a basic concept of the framework, complex structures are realizable. Since Sencha Touch also provides support for gestures and animations, it fulfills the criterion *User Interface Elements* very well (1). The look & feel of a Sencha Touch app resembles that of iOS apps. Sass (Sass, 2013), an extension to CSS, can be used for customization. Though Sass allows powerful and easy design adjustment, a *Native Look & Feel* would require high effort (4).

Sencha Touch provides a tool to merge and minimize source code and embed icons into CSS files. Thus, unused components are omitted and the number of HTTP requests is reduced. Additionally, the framework uses an update process similar to that of a native app by storing data in HTML5's application cache combined with versioning and incremental updates. However, an app can be as large as 800 KiB and, therefore, have a significant loading time. Summing up, the experienced *Load Time* on first launch can be rather high, but caching may reduce that on further launches (3). Runtime performance of a Sencha Touch application is close to native performance, as no lag or stuttering is observable (2).

5.5 Google Web Toolkit with mgwt

GWT and mgwt are open source and free of charge, also for commercial projects. Extensive support is also freely available, at least for GWT, so that the criterion *License and Costs* is fulfilled well (1). Since development is mainly sponsored by Google, GWT depends on Google's steady contribution to the framework. There have been some concerns about Google's long-term strategy with respect to GWT ("Comments on Google Web Toolkit Steering", 2012), but so far, it received continuous updates at least twice a year and its popularity, also within Google itself, should ensure its viability for the near future. mgwt, however, relies on a single free-lance developer and has no advanced project structure. So far, it has been updated regularly. Overall, *Long-term Feasibility*, especially of mgwt, has to be seen as uncertain (4).

While GWT's documentation is comprehensive and easily comprehensible, mgwt lacks a thorough documentation. Information is scattered among various resources, namely mgwt's wiki, blog, API documentation, and a showcase. mgwt lacks a tutorial that allows developers to learn how to develop with it. The Google Web Toolkit Community (2012) is large and offers support in a forum and on external sites. The community forum of mgwt (mgwt User Group, 2013) is also rather active, with approximately 70 topics per month over the last half year, and mgwt's developer answers questions on a daily basis. The active support partially offsets the mediocre documentation, so that *Documentation and Support* are satisfactory (3).

The initial steps with mgwt are rather hard, because a newly created project already contains more than ten classes and still needs additional ones to work properly. In this situation, the lack of a complete tutorial becomes especially noticeable. However, after mastering the initial hurdles, further familiarization is easy thanks to mgwt's clear structure and its easy-to-understand API. Typical Web developers need to learn Java, while, of course, GWT provides an advantage for Java developers. In summary, *Learning Success* is hindered at first but fast later (3).

Developing Web apps, even complex ones, with GWT is comparatively easy. Programming in Java enables developers to use first-class development environments such as Eclipse, which offer code completion and compile-time type checking. Hence, tool support is considerably better than it is for JavaScript programming, partly due to Java's static typing. Java's object-oriented nature offers sophisticated means for structuring the source code of an application. GWT imposes no restrictions on the architecture of applications, making it suitable for smaller and larger projects alike. At the same time, the framework provides a large set of often-needed features such as AJAX handling and internationalization. The API of mgwt is also directly accessible. Hence, implementing the UI programmatically is not as cumbersome as it is in other approaches. The development life cycle of GWT is slightly more complex, because an additional compilation of Java to JavaScript takes place. However, Google offers free plug-ins for Eclipse that handle these steps. Furthermore, extensive debugging facilities are available. In contrast, setting up new mgwt projects is complicated and requires several shell commands. All in all, the *Development Effort* is acceptable for all kinds of projects (2), especially for developers with experience in Java.

As the existence of mgwt itself demonstrates, GWT is extensible by everyone, mainly through object-oriented means. Google and third-parties provide plug-ins for a wide range of functionality such as maps and geo-location. *Extensibility* is good, although there is no formalized plug-in mechanism (2). Java source code of GWT- and mgwt-based apps lends itself well to a modular organization with separation of concerns. Although some boilerplate code is necessary, overall *Maintainability* is good (2).

The user interface is primarily influenced by mgwt, which provides mobile-optimized widgets and behavioral elements. The set of widgets provided by mgwt is limited and smaller than that of jQuery Mobile: It includes only standard buttons, one default list widget, and fewer form elements, most with a default appearance only. There are no advanced widgets such as map views. The behavior of widgets is noticeably optimized for mobile usage, for example with pull-to-refresh functionality for lists. Additionally, typical animations for page transition are available. All in all, mgwt provides good functionality with respect to mobile *User Interface Elements* (2), since the limited set of widgets is well-optimized.

mgwt has a particular focus on native appearance of Web apps. Therefore, it supplies CSS themes for different mobile platforms and devices, including iPhone, iPad, Android, and Blackberry. The widgets adapt to the respective platform and mimic the appearance of native UI elements as precise as possible. As developers may further change the appearance using CSS, mgwt achieves a *Native Look & Feel* (1).

The compiler of GWT tries to reduce the number of files to download and their size by removing unused code and optimizing the generated JavaScript. Hence, a mobile Web app with GWT and mgwt can be as small as 200 KiB. Additional support for HTML5's application cache ensures a fast *Load Time* (1). In addition to a native appearance, mgwt also tries to match

the feeling of native apps by focusing on a strong performance of the underlying JavaScript code. As a result, the *Runtime Performance* of mgwt apps is very good and in most cases close to native apps (1).

6 DISCUSSION

This section summarizes the strengths and weaknesses of each framework. We then analyze which framework tends to be best suited for typical scenarios. Eventually, we examine which promising approaches and which pitfalls currently exist, also outlining further need for research.

jQuery Mobile is a popular framework for building mobile user interfaces. It is easily accessible thanks to its documentation and by being based on HTML markup. By focusing on mobile UI elements, it neglects advanced requirements such as data binding or patterns for complex applications. Hence, smaller application with only little business logic can be created with ease, but larger applications do not benefit from jQuery Mobile. Integrating frameworks that provide such functionality is no simple task.

The-M-Project's JavaScript API supports such advanced requirements. Mobile Web apps are completely implemented in JavaScript. jQuery Mobile is only used in the background to create the UI and is not apparent to developers. The-M-Project is better suited for advanced requirements and large apps, at the cost of being less accessible at first. Extending it, also with respect to a native look & feel, is difficult. Load and runtime performance are average at best.

Web apps built with Sencha Touch are also developed solely in JavaScript. Its API is of a high quality, so that developers can learn and master the framework quickly. As the API is extensive and powerful, Sencha Touch is suitable for large and complex apps. These also benefit from good maintainability. For smaller apps with simple requirements, Sencha Touch's overhead may reduce efficiency of development.

Google Web Toolkit takes a different approach to Web development, as apps are written in Java and compiled to JavaScript. mgwt extends GWT with mobile-optimized widgets. Developing with GWT is easy, also due to good development environments. From a user's perspective, mgwt yields high-quality mobile Web apps with good performance and a native look & feel on Android and iOS. In contrast to GWT, mgwt's documentation and accessibility is below average. The long-term outlook is rather unstable.

These summaries already suggested some frameworks for certain requirements. The selection of a framework should be based on weighting the criteria according to their importance in the respective app project. Afterwards, the frameworks can be ranked with respect to their suitability for the project at hand. Table 3 provides a good starting point for the selection process. In general, small projects with only limited user interaction and logic tend to be well-supported by jQuery Mobile. As UI design is based on HTML markup, jQuery Mobile is easily accessible to teams skilled in Web development. An app with a restricted feature set suffers less from the lack of advanced functionality in jQuery Mobile. Full-fledged mobile Web apps with complex user interaction and advanced requirements are a different scenario. They need support for data binding, history management, modularity, and possibly internationalization. Sencha Touch is well-equipped for developing this kind of app.

If developers are used to Java, GWT might be a viable alternative, although it so far does not support mobile Web apps directly. mgwt adds this support, but is not as stable. If a look & feel is desired that to some extent resembles native apps, GWT and mgwt might also be suitable. However, in that case, a different approach to cross-platform mobile development than Web apps might be preferable. Which framework to choose also depends on the long-term importance of mobile Web apps. For a one-time development project, the adjustment time needed for a more complex framework such as Sencha Touch might not pay off, so that jQuery Mobile is a natural choice. In case of ongoing mobile development, using the more extensive frameworks might pay off.

Our evaluation highlighted promising approaches but also several areas of potential improvement. Weaknesses outlined in the individual evaluation sections are left up to the developers of each framework. Furthermore, we have identified several areas where additional research and exploration seem worthwhile. The examined frameworks either concentrate on building a mobile UI with HTML, not providing advanced functionality (jQuery Mobile), or implement UI and logic solely in JavaScript (The-M-Project, Sencha Touch). A combination of these approaches in the mobile context seems worthwhile, because it would allow developers to use well-tested technologies such as HTML and CSS to implement the UI, while resorting to JavaScript APIs for more complex logic. Such a combination would require means to connect mobile HTML components with corresponding JavaScript APIs, e. g., for data binding.

A better combination of technologies for native development and for Web apps is desirable. How future technology could combine the strengths of today's competing technologies is an open question. Research needs to deal not only with technology but

also with its application. To our knowledge, development best practices for mobile Web apps hardly exist. Further research needs to examine whether existing processes designed for Web development can be transferred and it should compile specific guidelines.

7 CONCLUSIONS

We presented an evaluation of frameworks for creating mobile Web apps. Based on typical requirements of apps, we derived a set of criteria that can be used to evaluate (and design) these frameworks. The criteria were applied to evaluate jQuery Mobile, The-M-Project, Sencha Touch and Google Web Toolkit combined with mgwt. Sencha Touch is deemed suitable for complex apps, while jQuery Mobile is sufficient in case of smaller projects focused on UI aspects.

While we designed our approach to be as objective as possible, some threats to validity remain: Each framework was evaluated by two of the authors as reviewers. To nevertheless ensure a diversified knowledge base, each reviewer brought a different area of expertise to the evaluation. They were experienced with software development in general as well as web and mobile development in particular. However, developers with yet another background might weigh the strengths and weaknesses of framework differently. We incorporated feedback from industry partners and community sources to address this issue. Furthermore, we examined one specific version of each framework. Thus, as quality changes over time, the individual assessment might not stay accurate. For the near future, however, it should hold. The general statements as part of the assessment will likely be true for an even longer period.

To overcome the issues mentioned above and to expand our research, future work includes enlisting more experts with diverse backgrounds for reviewing the frameworks, updating the evaluation to new versions, and addressing issues mentioned throughout the assessment and discussion. Moreover, we would like to give even more detailed decision advice.

REFERENCES

Apache License, Version 2.0. (2004). Retrieved Jan. 29, 2013, from http://www.apache.org/licenses/LICENSE-2.0.html

Basu, C., Poindexter, S., Drosen, J., and Addo, T. (2000). Diffusion of executive information systems in organizations and the shift to web technologies. *Indust. Manag. & Data Syst.*, 100:271–276.

Charland, A. and Leroux, B. (2011). Mobile application development: web vs. native. *Comm. ACM*, 54(5):49–53.

Chatterjee, D., Grewal, R., and Sambamurthy, V. (2002). Shaping up for e-commerce: institutional enablers of the organizational assimilation of web technologies. *MIS Q.*, 26(2):65–89.

"Comments on Google Web Toolkit Steering" (2012). Retrieved Jan. 29, 2013, from https://groups.google.com/d/topic/gwt-steering/qO9MW9 lSL5Y

Connors, A. and Sullivan, B. (2010). Mobile web application best practices. Technical report, W3C. Retrieved from http://www.w3.org/TR/mwabp/

Curran, K., Bond, A., and Fisher, G. (2012). HTML5 and the mobile web. *IJIDE*, 3(2).

Dougan, R. (2012). "Packaging Sencha Touch 2 with PhoneGap". Retrieved Jan. 29, 2013, from http://robertdougan.com/posts/packaging-sencha-touch-2-with-phonegap-cordova

Firtman, M. (2012). *jQuery Mobile: Up and Running*. O'Reilly, Sebastopol.

Fishburn, P. C. (1967). Additive utilities with incomplete product sets: Application to priorities and assignments. *Operations Research*, 15(3):537–542.

Gartner (2012). Gartner Press Release. Retrieved Jan. 29, 2013, from http://www.gartner.com/it/page.jsp?id=1924314

GNU General Public License (GPL). (2007). Retrieved Jan. 29, 2013, from http://www.gnu.org/licenses/gpl-3.0.html

Gong, J. and Tarasewich, P. (2004). Guidelines for handheld mobile device interface design. In *Proc. DSI 2004 Annual Meeting*.

Google Web Toolkit. (2012). Retrieved Jan. 29, 2013, from https://developers.google.com/web-toolkit/

Google Web Toolkit Community. (2012). Retrieved Jan. 29, 2013, from https://developers.google.com/web-toolkit/community

GWT-Phonegap. (2013). Retrieved Jan. 29, 2013, from http://code.google.com/p/gwt-phonegap/

GWT FAQ (2012). GWT FAQ. Get Started. Retrieved Jan. 29, 2013, from https://developers.google.com/web-toolkit/doc/latest/FAQ GettingStarted

Harjono, J., Ng, G., Kong, D., and Lo, J. (2010). Building smarter web applications with HTML5. In *Proc. CASCON '10*.

Heitkötter, H., Hanschke, S., and Majchrzak, T. A. (2012). Comparing cross-platform development approaches for mobile applications. In *Proc. 8th WEBIST*.

HTML5 (2012). HTML5. Retrieved Jan. 29, 2013, from http://www.w3.org/TR/html5/

jQuery. (2012). Retrieved Jan. 29, 2013, from http://jquery.com/

jQuery Mobile. (2013). Retrieved Jan. 29, 2013, from http://jquerymobile.com/

jQuery Mobile Forum. (2013). Retrieved Jan. 29, 2013, from http://forum.jquery.com/jquery-mobile

jQuery Project. (2012). Retrieved Jan. 29, 2013, from http://jquery.org/about/

Lennon, J. (2010). Compare JavaScript frameworks. Retrieved Jan. 29, 2013, from http://www.ibm.com/developerworks/java/library/wa-jsframeworks/

Lin, F. and Ye, W. (2009). Operating system battle in the ecosystem of smartphone industry. In *Proc. 2009 Int. Symp. on IEEC*.

Lubbers, P., Albers, B., and Salim, F. (2011). *Pro HTML5 Programming*. Apress, Berkeley.

"The-M-Docs. Native Packaging". (2012). Retrieved Jan. 29, 2013, from http://panacodalabs.github.com/The-M-Docs/#espresso/native packaging

Melamed, T. and Clayton, B. J. C. (2009). A Comparative Evaluation of HTML5 as a Pervasive Media Platform. In *Proc. 1st Int. ICST Conf. MobiCASE*.

Meyer, J. (2011). *HTML5 and JavaScript Projects*. Apress, Berkeley.

mgwt User Group (2013). mgwt User Group. Retrieved Jan. 29, 2013, from http://groups.google.com/group/mgwt

mgwt. (2013). Retrieved Jan. 29, 2013, from http://www.m-gwt.com/

The MIT License. (1988). Retrieved Jan. 29, 2013, from http://opensource.org/licenses/mit-license.php

The-M-Project. (2013). Retrieved Jan. 29, 2013, from http:// the-m-project.org/

"The-M-Project. github repository". (2013). Retrieved Jan. 29, 2013, from https://github.com/mwaylabs/The-M-Project

"The-M-Project. Google Groups". (2013). Retrieved Jan. 29, 2013, from https://groups.google.com/group/themproject

Oehlman, D. and Blanc, S. (2011). *Pro Android Web Apps*. Apress, Berkeley.

Ohrt, J., & Turau, V. (2012). Cross-platform development tools for smartphone applications. IEEE Computer, 45(9), 72–79.

Olaru, A. (2007). Selection Criteria for Javascript Frameworks. Retrieved Jan. 29, 2013, from http://www.infoq.com/ news/2007/12/choosing-javascript-frameworks

PhoneGap. (2013). Retrieved Jan. 29, 2013, from http://phonegap.com/

Rabin, J. and McCathieNevile, C. (2008). Mobile web best practices 1.0. Technical report, W3C. Retrieved from http://www.w3.org/TR/mobile-bp/

Saaty, T. (1986). Axiomatic foundation of the analytic hierarchy process. *Manag. Sci.*, 32(7):841–855.

Sass (2013). Sass. Sass. (2013). Retrieved Jan. 29, 2013, from http://sass-lang.com/

Sen, R., Subramaniam, C., and Nelson, M. L. (2008). Determinants of the choice of open source software license. *J. Manag. Inf. Syst.*, 25(3):207–240.

Sen, R., Subramaniam, C., and Nelson, M. L. (2011). Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between? *Decis. Support Syst.*, 52(1):199–206.

Sencha Architect (2013). Sencha Architect. Sencha Architect. (2013). Retrieved Jan. 29, 2013, from http://www.sencha.com/products/architect/

Sencha Touch. (2012). Retrieved Jan. 29, 2013, from http://www.sencha.com/products/touch/

Smutny, P. (2012). Mobile development tools and cross-platform solutions. In *Proc. 13th ICCC*.

"Stack Overflow. Tag jQuery Mobile" (2013). "Stack Overflow. Tag jQuery Mobile". Retrieved Jan. 29, 2013, from http://stackoverflow.com/questions/tagged/jquery-mobile

Tarasewich, P. (2003). Designing mobile commerce applications. *Comm. ACM*, 46(12):57–60.

Triantaphyllou, E. and Mann, S. H. (1989). An examination of the effectiveness of multi-dimensional decision-making methods: A decision-making paradox. *Decis. Support Syst.*, 5(3):303–312.

Walsh, D. (2008). *8 Considerations For Choosing Your Javascript Framework*. Retrieved Jan. 29, 2013, from http://css.dzone.com/news/8-considerations-choosing-your

Zibula, A., & Majchrzak, T. A. (2012). Developing a cross-platform mobile smart meter application using HTML5, jQuery Mobile and PhoneGap. In *Proc. 8th WEBIST*.