Experiences on Mobile Cross-Platform Application Development Using PhoneGap

Jussi Ronkainen, Juho Eskeli, Timo Urhemaa, Kaisa Koskela-Huotari
VTT Technical Research Centre of Finland
Finland
jussi.ronkainen@vtt.fi, juho.eskeli@vtt.fi, timo.urhemaa@vtt.fi, kaisa.koskela-huotari@vtt.fi

Abstract—**Cross-platform mobile application development frameworks are an attractive alternative to native application development, with potential for improved asset reuse and reduced development costs. Few reports exist, however, on determining their suitability for a given type of application or identifying their potential pitfalls. To address this, we report our experiences from implementing a hybrid web application demonstrator on Android, iOS, Windows Phone 8, and desktop platforms for cloud-based content sharing and co-creation. The hybrid web application approach was found adequate for implementing the demonstrator. Notable challenges discovered during the process were platform dependent variation in HTML5 feature support, differences in the way browsers interact with platform services, and lack of platform specific debugging tools. Based on the results, emphasis on debugging tool support is suggested, as well as early and frequent testing on all target platforms.**

Keywords-cross platform; multi platform; phonegap; jquery; cordova; cloud; cloud-based; content; content sharing; liquid experience

I. INTRODUCTION

The current mobile device market is dominated by two operating systems (Q4 2012: Android 69.7%, iOS 20.9%), and the global smartphone sales for 2013 is estimated to be close to one billion units [1]. In this light, cross-platform development approaches, which facilitate application development for multiple operating systems with a single code base, seem compelling. Furthermore, in the current market situation there could be some room for a third competitor (e.g., Windows Phone or BlackBerry) into the mix of operating systems, which could make cross- platform mobile application development even more lucrative for software developers.

The advantages of a cross-platform development approach compared to a multi-platform approach using native development platforms come from the use of a single codebase, which in turn can result in improved asset reuse and reduced development and maintenance costs, for example. Additionally, the barrier of entry into mobile application development can be lower in cross platform development environments where HTML, CSS, and JavaScript technologies are commonplace [2].

The downside of the cross-platform mobile development approach is that it may not be suitable in all situations, for example when native look and feel in user interface is required, or in games where adequate performance cannot be guaranteed [2].

For the reasons mentioned above we wanted to study the feasibility of the cross-platform approach for a specific application, and to learn of the potential pitfalls with the approach. As a result we want to share the experiences gained to practitioners in the field in form of practices that did or did not work.

To achieve this, we implemented a hybrid web application demo for cloud-based content sharing and co-creation. Our aim was to study the practicalities of cross-platform development on the popular PhoneGap platform to gain an understanding of its strengths and weaknesses, as well as the skills and effort required. As a secondary objective, we studied the suitability of a hybrid web application approach for our particular application.

In the next section, the application concept is explained. Section 3 illustrates our implementation approach, along with expected results. Section 4 discusses mobile cross-platform development approaches with respect to identified state of the art. Results are described in Section 5, followed by conclusions and future work in Section 6.

II. CASE CONTEXT

The background for developing the application is in our previous research into the way people understand digital content, how they currently use it, and how they would want to use it. Sixty people participated in the research via the online user interaction forum Owela [3]. Of the 71 narratives and more than a thousand discussion comments provided by the participants, we chose photographs as the theme for our application.

We wanted to focus on the ease of content sharing and co-creation because of their perceived importance in many of the user stories. Cloud storage for the photos was also a recurring theme in the stories, and an evident requirement also because the wider context of the Cloud Sofware Program in which this research was carried out.

Our earlier research in content sharing and co-creation had also focused on the concept of liquid experience [4], which aims to provide users with a consistent experience regardless of the device used for accessing the information. This concept also gave us more freedom in choosing the cross-platform framework since following native application look and feel on each device platform was not deemed critical.

III. APPROACH

Wishing to experiment further with the liquid experience concept, we chose to implement the application as a hybrid web application that would allow running it standalone on Android, iOS, and Windows Phone 8 and, with some restrictions, on a desktop browser. For backend, we chose to use Google App Engine mostly because of our prior experience with it, and because it offers rudimentary image manipulation functionality we assumed could be useful. At a later stage in the project, we also evaluated the feasibility of porting the application to another backend, experiences of which will be briefly discussed later.

A. Framework Selection

We didn't want to limit the application to any particular platform or device. At the time of writing, HTML5 based approaches support the most platforms and also, via the use of CSS3, make adaptation to different screen sizes and orientations relatively simple. Of the available HTML5 cross-platform frameworks, we chose PhoneGap due to its widest device support and because it imposes minimal restrictions to applications that utilize it. Also, PhoneGap enables the packaging of HTML5 applications as native applications. PhoneGap ships without visual components, which makes it very flexible in terms of UI, but also means that the developer has to choose or implement all application components oneself and ensure they will work together. Browser based applications also have a performance overhead with respect to native applications but that was not considered an issue, since the performance requirements for our application were considered very modest.

Furthermore, the PhoneGap framework has a plugin interface for running native code that can access device capabilities. Many common plugins such as GPS, camera and local file access are implemented by default in PhoneGap. Utilizing these plugins does not require any native development skills. PhoneGap also supports custom plugins, so the application can be extended to use native code for functionality that is not supported in HTML5 or PhoneGap by default, or which would be computationally too intensive to implement in JavaScript. Although our application does not make much use of PhoneGap plugins, from a research perspective we found native code support to be an important feature in cross-platform development for added flexibility.

From application development point of view, there are many JavaScript frameworks available that focus on, for example, the graphical user interface and widgets, DOM tree manipulation, and web application architecture (Model-view-controller).

We chose the jQuery Mobile application framework as the JavaScript library for implementing the application UI. jQuery was used for DOM tree manipulation and Ajax based server requests. Our choice of frameworks was largely influenced by the vast popularity jQuery, and in case of jQuery Mobile, the fact that it seemed to provide wider platform support than most similar frameworks. Both jQuery Mobile and PhoneGap have active user communities and both projects are frequently updated and well documented. In addition, many examples and demos paved our way to choose these platforms.

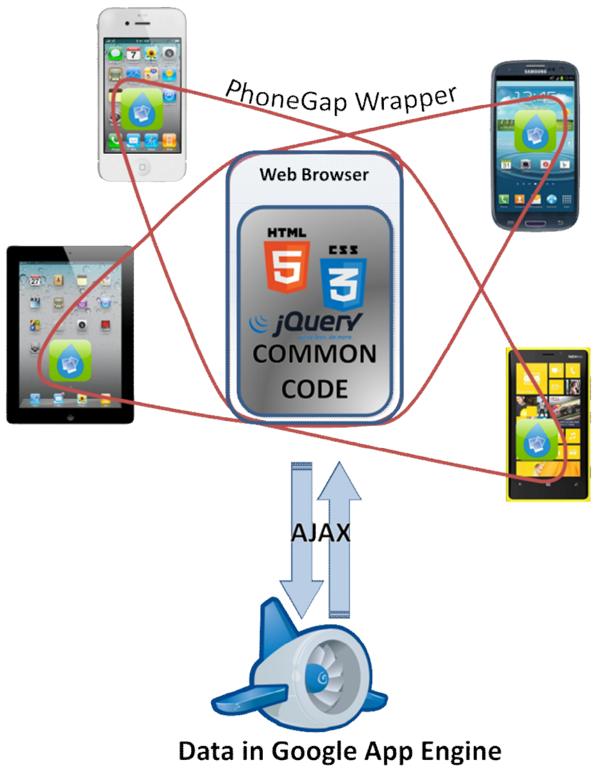

Figure 1. Our client/server structure.

B. Development Methods and Tools

We had three developers, each focusing on one platform in particular and the common codebase in general. This gave us an opportunity to observe the multi-platform development procedures with respect to, e.g., version control where the common application codebase had to be integrated into three platform specific codebases.

Our version control setup was such that there was a Git repository for each native project, and a repository for the common application code. The common repository was included into each native project as a Git submodule. Implementation was done on platform specific preferred editors for the native part - Xcode for iOS, Eclipse for Android, and Visual Studio for Windows Phone 8. HTML5, JavaScript and CSS3 editing was mostly done with JetBrains WebStorm.

The testing, largely UI driven and ad hoc, was done by the developers themselves on desktop browsers, mobile devices, and device emulators. Some functionality was also tested as automated unit tests on the Jasmine JavaScript unit test environment, which was run on a desktop browser.

C. Expected Results

From user interface point of view, we expected to have to make some conditional layout in the common code to cater for different screen sizes and resolutions. Also, we expected some minor variations in the way the application would render on the different devices. But for the most part, we assumed the ”write once, run anywhere” promise of cross-platform development to work more or less straight out of the box, especially since we were using the popular jQuery Mobile framework which we assumed to be well adapted for most platforms.

PhoneGap uses the device's browser as the application platform. Browsers are complex applications themselves, meaning that there is performance overhead for applications running on them. Also, since browsers have to handle all kinds of content, they are not optimized for any specific kind of application. Different browsers support HTML5 features to varying degrees, so application performance on different platforms might also vary. Performance was not, however, considered to be an issue in our case due to the simplicity of the application.

The use of HTML5 and jQuery / Ajax as the common implementation technology on all platforms was also expected to make cloud resource access simple.

IV. Related Work

We identified the current state of the art in mobile cross-platform development ([2][5][6][7][8][9]). In the following, cross-platform approaches are described in general, followed by a detailed discussion on one publication which most closely relates to our work discussed in more detail.

In *native application development* approach the application is implemented for a particular platform (as opposed to multiple platforms in cross-platform development) by using the provided Software Development Kit (SDK). The applications developed in this fashion maintain the look and feel of the platform. Porting the application to another platform is not possible without additional effort.

We consider *cross platform application development* to be development that is done with the help of cross-platform framework or with combination of platforms. Combining of platforms may be required because the frameworks focus on different purposes; some of them support development of complete applications that include application logic, user interface, and deployment, while some of them may focus on just one of these [2]. Related to UI representation in frameworks there are two different approaches commonly used; to imitate native look and feel (or use native components), or to maintain uniform look and feel for the supported platforms that ignores the native styling [2].

A definition of cross-platform frameworks is given by Sommer as follows: "Cross-platform frameworks are frameworks that support multiple platforms, with the same or similar effort involved to create an application on potentially more than one platform at once (or porting an application to other platforms with very little effort), as compared to creating it for only one platform with the native SDK. This essentially requires that a framework has to provide means to reuse parts of the architecture and source code that are platform-independent" [2].

The most commonly used frameworks in mobile cross-platform development can be categorized by the architectural approach taken into web-based, hybrid, and self-contained categories as presented in [2][6][7]. The publications also mention other types of approaches that utilize, e.g., cross compilation techniques. However, none of the current cross-compilation solutions that we are aware of are ready for production quality application deployments to prevalent mobile operating systems (e.g., Qt Alpha 5.1 advertises preliminary support for Android and iOS, with full support announced later in the oncoming 5.2 version).

By utilizing *web based frameworks* the application is developed as regular web site using HTML, CSS, and JavaScript technologies. An example framework in this category is jQuery Mobile. Pure web applications cannot be installed in similar fashion as native applications nor can they access the sensors or actuators of the mobile device.

In *hybrid frameworks,* the web based and native approach have been combined to create applications that inherit features of native applications (e.g., capability to install from an application store, native fashion application launching, capability to interface with sensors and actuators) but are developed using web technologies. An example framework from this category is PhoneGap.

Self-contained runtime environments, as described in [2], do not attempt to reuse existing web frameworks of the selected platform. By implementing their own web container the frameworks are in theory less constrained by any shortcomings in platform frameworks. Example of this type of framework is Titanium Mobile.

Zibula and Majchrzak [9] document the development of a Smart Metering Application using similar tool set as in our work. They outline the relevance of continuous testing on all target platforms because bugs might be visible only on a single platform. Our experiences also highlight the importance of continuous testing in cross-platform development. They also mention immaturity of the frameworks used, namely jQuery Mobile. We didn't face as severe problems in our work, which could be an indication that the frameworks have matured already. They also mention the debugging tools that they used, but don't go into detailed discussion about debugging, other than that the tools were very useful. Based on our experience debugging is one of the more important issues in cross-platform mobile development in which we focus in more detail in our work. Finally, they note that the hybrid approach is viable and advisable approach for cross platform development, but that in the long term it could be a transitional technology that may be replaced by pure HTML5 approach. While this may turn out to be true, we think that some form of tool or a solution is still needed to wrap the HTML5 application as a native application, and additionally HTML5 is unlikely to allow native extensions for whatever purpose. Zibula and Majchrzak also note that usability (of cross-platform developed mobile applications) and value for users are important research topics to consider besides technological development.

V. Results

Overall, we feel the application demo we implemented is complex enough to get an idea of the potential of hybrid web applications and to gather meaningful experiences from building it. Figure 2. shows a screenshot of the application during photo sharing on Android, Windows Phone and iOS devices. The figure illustrates differences due to the different fonts and screen aspect ratios on the devices.

We have divided our findings into three main groups; platform specific findings, user interface findings, and findings on the development process in general.

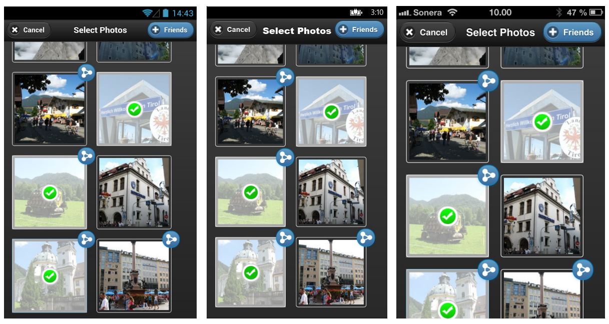

Figure 2. User photo sharing screen on Android (Galaxy Nexus, left), WP8 (Nokia Lumia 920, middle), and iOS (iPhone 4, right).

A. *User Interface Findings*

We found the user interface rendered from the common codebase to be fairly consistent among the platforms. This was largely due to our use of the jQuery Mobile framework which provided most of the UI elements. On the phones we tested, there were some nuances caused by different default fonts and different screen aspects, as illustrated in Figure 2. We used seven CSS3 media queries to set UI component dimensions to cater for all the screen sizes and orientations on the phones and tablets we had. In general, we found the underlying browser engines to do a good job in laying out the application on different screens and orientations. Some layout issues were discovered, such as different default page footer element handling on WebKit based vs. Windows Phone 8 devices but these could be fixed with platform specific style definitions.

We also encountered a few UI issues that affected only some platforms, such as page transition animations flickering on Android and completely missing or visually different on Windows Phone 8, and difficulties in disabling the default visual cue when attempting to scroll past the end of page on Windows Phone 8. Some of these issues have already been fixed in recent jQuery Mobile and PhoneGap versions, and we assume such easily noticeable visual differences will be fixed in future versions. However, we had to use platform specific style definitions from time to time to enable, e.g., HW acceleration for UI transition effects.

Another source of UI issues was the virtual keyboard which is unique to each platform. The screen area taken by the keyboard varies, as does its interaction with the underlying application. In our experience, the effect of the virtual keyboard needs to be tested thoroughly on each platform.

Probably the most notable issue we discovered, however, was the occasional sluggishness of touch input. This seemed to affect all platforms at some time or another. Most commonly there were missed touch events such as pressing a button or starting a swipe. The issues were random and slight but still noticeable and detrimental to a smooth user experience. We did not analyze the cause of the sluggishness but to get the UI really responsive would probably require platform specific analysis and optimization of the HTML5/JavaScript/CSS3 code. Also, we did not pay any attention to DOM tree optimization, which at least in large applications could have a significant effect in application performance.

In general, UI event support was found to differ between browsers and if mobile and desktop browsers are to be supported, both touch and mouse events need to be handled. Also, touch event support differs between platforms – for example, not all jQuery Mobile swipe events work on Windows Phone 8 without platform specific HTML5 style definitions. For this reason it is necessary to test all UI events as early as possible on all devices, and support multiple navigation methods where possible.

B. *Platform Specific Findings*

In addition to user interface issues which were caused by the differences in browser rendering engines, there were a couple of platform specific issues we could not solve or circumvent by modifying the application.

By request from a Cloud Software Program partner, we briefly experimented with the possibility of porting the application to use another backend. During our trials with the second backend which used HTTPS we came across a problem with SSL certificates. The development installation of the backend used a static IP address without a domain name, which meant that browsers could not ensure the authenticity of the certificate. On desktop browsers we could add an exception, and on the Android PhoneGap version we observed no issues. However, we could not get iPhone to create an exception for the server. This meant that the iOS application could not be run against that backend. While the problem is eliminated when the certificate is tied to a domain name, it could be a problem during development as in our case. Certificate handling was not tested using Windows Phone 8.

Another issue we could not solve from within the application was with browser cookies on Windows Phone 8. Our application uses a session cookie received from the server at login to identify the user during subsequent operations. PhoneGap obtains the cookie settings from the browser, but these settings vary between platforms. On Windows Phone 8, we had to change the system wide cookie settings manually on the browser of the device in order to get the application to store the session cookie. This, of course, is not acceptable for a consumer application. The need for cookies could be averted by implementing an authentication token scheme on the client and the server but that would require extra work.

Overall, however, we found cloud-based resource access straightforward and uniform across all platforms.

Native plugins are also a source of platform specific differences. It should be noted that even the plugins that ship with PhoneGap are not supported on all platforms, so the need for native support should be considered early on in a cross-platform development project. We implemented a native application settings screen on each platform and passed the settings to the HTML application via the plugin interface. Activation of the settings screen was also done via the interface. We found the plugin interface to work quite well. The native side of the plugins can be debugged on platform specific development environments like any native code.

C. *Development Method and Tool Findings*

JavaScript is an interpreted language, meaning that without a compiler, the role of the editor in finding programming errors is emphasized.

While all native development environments (Xcode, Eclipse, Visual Studio) support the development of HTML5 applications, none of them in our opinion match the best of dedicated HTML5 editors. Also, the use of a common editor for the HTML5 application by all developers in a project is justifiable in order to establish, e.g., common practices and file templates. While significant parts of an application can be implemented against a desktop web browser, deploying the application on a device, however, requires the native development environment. This causes extra steps and switching between applications in the development process.

We found automated unit testing useful in detecting problems in program logic earlier. Running unit tests with a framework such as Jasmine is quick and isolates program logic issues well. We ran a limited set of unit tests on a desktop browser and because of the ease of running the test suite, unit testing was useful in detecting programming errors quickly. Unit testing frameworks typically provide means for writing stubs, spies and mocks that enable the separation of, e.g., network code from the UI. This helps in isolating program logic issues and programming errors, but in our experience, automated unit testing frameworks are of limited use in exposing issues related to the target platform.

We also found the SW project structure to have significance in cross-platform development. Since in our case the common application code project was included as a subproject in each of the native projects, we occasionally ended up with subproject version conflicts. In the Git version control system the only links between the main repository and the submodules are submodule IDs which are saved in the main repository, and in some situations changes in the IDs are not automatically reflected into the submodules. As a result, we ended up cloning the common module as a separate project into the appropriate directory in each native project, and excluding the directory from version control in the native projects. Automatic refreshing of the subproject during native project refresh was thus lost, but in our case extra manual work caused by that was negligible since the native projects were changed much less frequently than the common project. Native project updates were mostly PhoneGap version updates. In our experience, however, they need to be done with care as PhoneGap version updates usually have to be synchronized between all native projects and the common project. Occasionally, a new PhoneGap version forced us to recreate the native projects from scratch. The documentation of the new release was also outdated at times, which caused some extra work to solve out the native project upgrade process.

To reduce the need for handling native projects, Adobe offers the cloud-based PhoneGap Build service which builds native applications from the HTML5, JavaScript and CSS code. There are, however, restrictions to custom plugins in PhoneGap Build.

The most significant shortcoming we experienced during development was the limited debugging ability of PhoneGap applications. The reason is that the embedded native browser PhoneGap uses is not accessible to a debugger on every platform, and thus problems that arise only on a specific platform may be very difficult to debug. At the time of writing, only BlackBerry and iOS browsers offer remote debugging that can be extended to PhoneGap applications. The Chrome browser on Android offers remote debugging but not via PhoneGap. Windows Phone 8 lacks remote debugging capability for both of the scenarios. At the time of writing, the best solution for remote debugging of hybrid web applications is Apple's development tools for iOS. Xcode in combination with Safari on Mac offers all required debugging capabilities including DOM tree manipulation, breakpoints and variable inspector.

For most of the time we used a desktop browser for debugging, occasionally augmented by the PhoneGap Emulator on Google Chrome. The emulator was useful in verifying the UI with different screen sizes and resolutions, and getting a hang of using the native interfaces exposed by PhoneGap, although the emulator mostly uses mock data for them. A good rule of thumb for hybrid web application development is to use desktop browsers so that Chrome is used as a preliminary test for Android, Safari for iOS and IE for Windows Phone. Some browsers also have built-in tools for simulating different mobile device screen sizes.

Another useful PhoneGap debugging tool we used is weinre that is available either as a local installation or online via debug.phonegap.com. While weinre does not offer breakpoints, it does allow the inspection, highlighting and modification of DOM elements and JavaScript variables via a console.

PhoneGap can also relay the JavaScript console.log() output to the development environment console window. We found debug prints to console a viable debugging method, although understandably limited.

D. *Summary of Findings*

HTML5-based cross-platform applications rely heavily on the web browser on each platform, and differences in how the browsers implement HTML5 features were the underlying cause for most of our findings. In particular, we found occasional platform specific issues with page element layout and certain jQuery Mobile page animations, and touch event support. Most issues were solved by platform specific code and style definitions, but the intermittent problems with touch input responsiveness on all platforms were not.

Issues were also encountered in the way the browsers interact with their surroundings, namely in the visual cue the browsers give on trying to scroll past page boundaries, virtual keyboard behaviour, SSL certificate handling, cookie handling, and PhoneGap plugin support. While some of the issues were remedied via native project settings, solutions were not found during this study for the SSL certificate and cookie problems.

From a developer viewpoint, we found a dedicated HTML editor more useful than native IDEs which are typically not optimized for editing HTML5. Support for debugging on the device is only possible on iOS and Blackberry at the moment, which was found to be the biggest drawback of the approach. When device debugging is not required, desktop browsers provide good debugging options – although their use is not as seamless as debuggers on native IDEs.

VI. DISCUSSION

In our experiment, we implemented a content sharing and co-creation application using PhoneGap and jQuery Mobile. We found the approach to fit our type of application well, and platform specific additions to the common codebase to be fairly minimal. HTML5 and CSS3 were found to do an efficient job of scaling the layout to different screen sizes and orientations, and that in general, the UI renders smoothly on the different platforms. However, we encountered issues with jQuery Mobile animations, so it is advisable to keep them to a minimum. This is particularly important if the targeted range of platforms is wide, or targeted devices are of modest performance or use old web browser engines.

There were also issues with UI responsiveness. Some issues we were able to fix via platform specific, non-standard style definitions, but we could not quite reach consistent, native quality responsiveness on any of the platforms.

Development tools were found adequate for most of the time, when the code could be developed and tested against a desktop browser. Automated unit testing was also experimented, and found useful in finding program logic bugs quickly.

Debugging on the target devices is the area that is in our experience most evidently lacking in hybrid web application development. The role of debugging is emphasized by the loosely typed, interpreted nature of Javascript, as without a compiler there are fewer safety nets to catch programming errors early. For limited device debugging we experimented with weinre and the PhoneGap emulator. Both were found useful, but lacking in functionality. Problems that do not surface on a desktop browser tend to concern non-standard HTML5 / CSS3 extensions or other platform specific browser behaviour. Thus, solving these problems is difficult without platform specific source-level debugging with breakpoints. For these reasons, the role of active and early testing on every platform is paramount.

VII. CONCLUSIONS AND FUTURE WORK

The current smartphone and tablet market has made it necessary to develop applications for several platforms. Cross-platform development approaches are one way of increasing asset reuse between platforms and reducing development cost. Our study focused on the hybrid web application approach using the popular PhoneGap platform.

Overall, the approach was found solid and suitable for the type of application presented in the study. The biggest drawback encountered in the approach is insufficient debugging support on mobile devices. Platform specific variation in HTML5 feature support and browser interaction with the platform were found to necessitate constant testing on all platforms. UI performance issues that varied between mobile platforms were also encountered. Examining them would be one potential objective for future research.

Comparison of the hybrid web application approach with other cross-platform approaches would be another interesting topic, perhaps by implementing the same demonstrator using different approaches.

ACKNOWLEDGMENT

The authors wish to thank the TiViT Cloud Software Program in which this research has been carried out.

REFERENCES

[1] Gartner, “Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012”, Press release, http://www.gartner.com/newsroom/id/2335616 08.08.2013

[2] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, “Evaluating Cross-Platform Development Approaches for Mobile Applications,” in Lecture Notes in Business Information Processing, Volume 140, 2013, pp. 120-138

[3] P. Näkki and K. Koskela-Huotari, “User Participation in Software Design via Social Media: Experiences from a Case Study with Consumers,” in AIS Transactions on Human-Computer Interaction, vol. 4, 2012, pp. 128-151.

[4] H. Kiljander and V. Nore, “Experiences from Long-Term Online User Collaboration in Strategic Product Design,” in Proceedings of NordiCHI 2012, Industrial Track, ACM.

[5] A. Sommer, Comparison and evaluation of cross-platform frameworks for the development of mobile business applications, Master’s thesis, Fakultät für Informatik, Technische Universität München, 2012.

[6] A. Holzinger, P. Treitler, and W. Slany, “Making Apps Useable on Multiple Different Mobile Platform: On Interoperability for Business Application Development on Smartphones,” in Multidisciplinary Research and Practive for Information Systems, Lecture Notes in Computer Science, Volume 7465, 2012, pp. 176-189.

[7] E. Masi, G. Cantone, M. Mastrofini, G. Calavaro, and P. Subiaco, ”Mobile Apps Development: A Framework for Technology Decision Making,” in Mobile Computing, Applications, and Services. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Volume 110, 2013, pp. 64-79.

[8] L. Corral, A. Janes, and T. Remencius, “Potential Advantages and Disadvantages of Multiplatform Development Frameworks – A Vision on Mobile Environments,” in Procedia Computer Science, Volume 10, 2012, pp. 1202-1207.

[9] A. Zibula and T. A. Majchrzak, “Cross-Platform Development Using HTML5, jQuery Mobile, and PhoneGap: Realizing a Smart Meter Application,” in Web Information Systems and Technologies, Lecture Notes in Business Information Processing, Volume 140, 2013, pp. 16-33.