Selecting a Mobile Platform Strategy

Robert S. Bauer

***Abstract*—This research evaluates product-line software engineering of mobile applications. A native mobile application benefits from performance and device integration, but at the cost of multiple versions of the application for each mobile platform, e.g., an Android and iPhone version. Engineering requirements may dictate a native application but alternatives are available which may allow a write-once-deploy-anywhere solution, reducing development and maintenance costs and requiring a smaller set of skills. This research shows HTML5 and the mobile web may be a valid solution for most applications if the application requirements do not rely on device optimization or hardware features.**

I. Introduction

The landscape for mobile devices has gone from feature phones to smartphones, with smartphones providing more CPU power and storage, allowing users to run applications of their choosing. To meet the demand for smart phones, there are several companies providing smartphone solutions, such as Apple (iOS), Google (Android), Microsoft (Windows Mobile), and others. Selecting a single platform to develop and release mobile applications on is no longer a viable option if the goal is to reach the most customers. See Fig. 1 for market share breakdown by platform.

Platform	Market Share (%)
Google	50.1%
Apple	30.2%
RIM	13.4%
Microsoft	3.9%
Symbian	1.5%

Fig. 1. Platform market share for February 2012 [1].

Each platform typically has its own software development kit and language or languages supported. Each platform has its own capabilities and tool sets. Developing a native application to support each platform would be difficult unless the developer has the necessary skill sets, for example, an application developed for one platform does not easily translate to another platform, see Fig. 2.

A native application has the benefit of accessing the device's APIs and frameworks, making the best use of the device's features. This requires a developer's specialization with the device's hardware and software stack to get the most out of the device. This makes native more expensive to build for each platform.

By abstracting the details on how to interact with the device, a developer may not need this level of detailed knowledge of the device. Although abstracting out the detail may restrict the capabilities of the application.

Mobile OS Type	Skill Set Required
Apple iOS	Objective-C
Google Android	Java
RIM BlackBerry	Java
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile, 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

Fig. 2. Required skill sets for mobile OS [2].

There are alternatives to developing native applications. These alternatives attempt to abstract commonalities between the devices at different layers. For example, all smartphones have a web browser [2]. A mobile web application may be an alternative. Another alternative is a hybrid approach which uses a framework to embed the device's browser in the application and provides application programming interfaces (APIs) to allow web code to interact with the device hardware. There are several approaches for developing mobile applications, each with its benefits and disadvantages as well as levels of software engineering reuse.

Mobile web applications, especially those leveraging the features of HTML5, have potential to resolve the issues writing native applications. For example, mobile web benefits for a large install base, good distribution, and is developer supported (most developers know HTML and JavaScript) [3]. Another benefit is the mobile web application does not have to give up a percentage of its fee back to the app store it is housed in, such as the 30% Apple App Store fee [5]. HTML5 APIs include the ability to interact with the application in online and offline mode, developers can persist the data on the client using SQL, and APIs for audio, video, and limited device sensors, such as GPS.

Mobile web sounds like an ideal solution, but has downsides. There is limited support for non-location device sensors. Content capture by camera and microphone is limited. The interruption of the user is limited to alerts and the application can not execute as a background task and provide notifications other than audio.

Another option attempts to combine the benefits of both native code and web development. It is considered a hybrid and provides a cross platform development framework that turns the mobile application into an embedded browser and provides APIs for JavaScript. The application is then developed using HTML5, CSS, and JavaScript. Frameworks available to do this are RhoMobile's Rhodes, MoSync, and the

open source solution, PhoneGap. These APIs allow JavaScript access to the devices features and sensors and attempts to mimic the device's user interface look and feel. As stated by J. Dehlinger and J. Dixon, "HTML5 tools, like PhoneGap, try to create a near native application for multiple platforms, but this does not allow for rich features that have access to the device's APIs and is a technological solution rather than a software engineering solution that allows reuse of engineering assets."

This research evaluates the alternatives to native application development and the software engineering issues.

II. Results

Anthony Wasserman surveyed mobile developers to get an understanding of their engineering practices and found the following [6]:

- Most apps were small
- One to two developers working on the same app
- Sharp divide between native and web
- Developers adhere to best practices but rarely use formal development processes
- Developers rarely track their effort or took metrics

Developers are following software engineering best practices for mobile development, but J. Dehlinger and J. Dixon suggest more could be done regarding mobile software engineering. They suggest mobile should have hierarchical structures consisting of the following [7]:

- Business layer
- Software development kit layer (SDK)
- Hardware-dependent layer (HdS)
- Hardware layer (execution platform)

Each layer would be positioned above the next in the mentioned order. For mobile, each layer is variable and changes. The current state of mobile development "is still monolithic" [7]. The authors state an "actual hierarchy" in the development process is necessary for mobile development.

By doing so, this would allow for two categories of mobile designers. One would be a software platform provider, which would implement services provided by the hardware and SDK. The software platform provider would also provide abstractions needed for the application developers to use. PhoneGap and Rhodes assume the application developer has strong programming skills and instead, this activity should be taken care of by the software platform provider.

The second mobile designer would be the application developer. Their roll would be to implement the user interface and user experience with a focus on the business and end user logic. They shouldn't have to worry about cross device compatibility. They would be dependant on the APIs developed and provided by the software platform provider.

By splitting the development rolls, software engineering experience and skill can be directed. The experienced mobile developer can abstract the hardware and SDK for the application developer to work with. The process of doing so abstracts and modularizes mobile development.

At this time, this approach has some challenges. The appropriate abstractions for each development layer needs to be defined. "Synthesis processes capable of transforming the semantics of the end-user application into cost-effective runtime code while taking account the non-functional requirements of the application and of the execution platform" need to be defined. Support needs to be added for application testing and fault-tolerance, as well as handling unidentified faults in the platform. Be able to leverage hardware-related issues, such as the availability of multi-core devices and distributed computing environments. Lastly, support for system evolution for the execution platform, requirements, and programming languages.

A development hierarchy for mobile development would be ideal. In its place, there are cross platform frameworks available, although as mentioned, these are a technical solution instead of a software engineering solution. Developing a mobile website is feasible too, although it has its limitations and may not be an ideal solution depending on the application's requirements.

If a native approach is still desired and multiple platforms need to be supported, a mobile application software product line should be considered. The product line would be a set of applications that share the same core requirements, yet are different according to a set of variable requirements [4]. This approach can reduce software engineering time and cost. A domain engineering phase would define the requirements for both the common and variable aspects of the entire product line. The application engineering phase would reuse these to develop the specific applications within the product line. By forcing developers to think about the common requirements, design, and resources, this would benefit mobile development. The author has participated in an Android and iOS development project which used and benefited from this technique.

III. Conclusion

Which platform to develop for? There are some options and the final decision would have to be based on the application's needs, features, and capabilities.

Develop for a single platform and use a subset of features. For example, develop for the iPhone, iPad, and iPod, but use only the features available to all of these devices.

Develop native applications for each platform. The trade-off is higher development and maintenance. But in return, the application is optimized for performance and functionality.

Develop mobile web application to develop once and use across multiple platforms and devices. It is uncertain if mobile web applications will meet the needs of the market [6]. It is critical to identify device sensors to be incorporated into the application and determine if HTML5 APIs are support in the device browsers.

Develop using one or more layered abstractions (hybrid) that can map a write-once application to a native application that runs on multiple platforms. This solution may be the best fit for those looking to support multiple platforms and devices.

REFERENCES

[1] comScore. (2012, April 7) "comScore Reports February 2012 U.S. Mobile Subscriber Market Share," [Online]. Available: http://www.comscore.com/Press_Events/Press_Releases/2012/4/comScore_Reports_February_2012_U.S._Mobile_Subscriber_Market_Share

[2] Andre Charland and Brian Leroux. "Mobile application development: web vs. native." *Commun. ACM* 54, 5 (May 2011), 49-53. DOI=10.1145/1941487.1941504 [Online]. Available: http://doi.acm.org/10.1145/1941487.1941504

[3] T. Melamed, B. Clayton, "A Comparative Evaluation of HTML5 as a Pervasive Media Platform," Springer Berlin Heidelberg, 2010, pp. 307-325

[4] J. Dehlinger, J. Dixon, "Mobile Application Software Engineering: Challenges and Research Directions." *Workshop Papers*. (October 2011). [Online]. Available: http://www.mobileseworkshop.org/papers/7-Dehlinger_Dixon.pdf

[5] K. Jordan, "Is HTML5 the Solution to App Store Fees?" *Editor & Publisher*. N.p.,. 2011. [Online]. Available: http://www.editorandpublisher.com/Features/Article/Is-HTML5-the-Solution-to-App-Store-Fees-

[6] A. Wasserman. "Software Engineering Issues for Mobile Application Development." *Workshop Papers*. 2011. [Online]. Available: http://www.mobileseworkshop.org/papers/Wasserman_foser2010.pdf

[7] J. Dehlinger, J. Dixon, "XModel: an Unified Effort Towards the Development of High-Quality Mobile Applications." *Workshop Papers*. (October 2011). [Online]. Available: http://www.mobileseworkshop.org/papers/8-Cota_etal_UFRGS.pdf