Chapter 1

Introduction

Imagine a smaller company with a well developed web application. The company has a few developers and has the want to extend their web applications features for their users. They want to track their users activity for making a health profile. To track the user's activity they need to use the mobile accelerometer. The accelerometer can be accessed through a mobile application but not through a web application.

The company can choose to develop a mobile application from the ground up and integrate the application with their current web application. That will cost them a huge amount of development time and also a lot of time maintaining several platforms.

Another less costly alternative is to create the mobile application using the existing web application. The web application is run within a shell of the mobile application. The web application can then access the mobile's native functions such as the accelerometer. This will save the company time developing the mobile application and instead of maintaining several platforms for every feature the mobile application only need to be maintained in regards of the use of the accelerometer.

The aim of this thesis is to evaluate two different development methods for encapsulating an existing web application in a mobile application to utilize native functions. With focus on evaluating development effort.

1.1 Motivation

There are a number of researches of developing a native or hybrid mobile application using different methods. The researches focuses on aspects such as cost, effort and advantages and disadvantages for developing an entirely new mobile application. However, it is very hard to find research on extending an existing web application with mobile native features.

The cost for developing a small to medium sized project in native Android or PhoneGap is estimated to cost around 20 000 - 40 000 dollars[3].

One of the huge advantages developing in PhoneGap compared with developing natively is that PhoneGap can be compiled to multiple platforms. Making PhoneGap an attractive development framework for companies with less development resources. If a mobile application is developed natively for Android, IOs and Windows Phone, maintaining the mobile application must be done in three different native applications. If the mobile application is instead developed in PhoneGap only one application needs to be maintained.

A disadvantage of developing in PhoneGap is when their is a use of many native features. PhoneGap relies on a development framework and the provided features when building a mobile application. Hence, if the framework is not up to date with the latest new features, the developer will not be able to partake of the features until the framework is updated. If the application is dependent on many native features a hybrid application may have limitations[3].

1.2 Problem formulation

The goal of this project is to evaluate two different development methods for encapsulating an existing web application in an Android mobile application. When evaluating the methods the focus is on development effort. The purpose of the mobile application is to give the web application access to the mobile's native functions. An example of such a native function would be accessing the mobile's accelerometer. It is important that the logic of the web application can be written in a general way (non-platform dependant) adapting to the device accessing the web application. So that the same web application code is used for the web application and mobile application.

Antoher goal is that the mobile application only provides data from native functions. In order to achieve that the web application and mobile application should have a master slave relationship. Where the web application acts as the master and the mobile application as the slave. To get data from the mobiles native functions the web application asks for the data and the mobile application passes the data back.

To enable this master and slave relationship the mobile and web application layer must be able to communicate. Passing commands and data between the layers. It is important the mobile and web application layers has a way of communicating that is developer friendly.

To limit the scope of the research the development methods is only evaluted for the mobile operating system Android. Android was choosen since it is the most widely used today for mobiles.

There are two development methods of creating an Android application that will be evaluated. The first development method is to develop natively in Android. The second method is to develop using the framework PhoneGap.

The methods will be evaluated and compared from a developers perspective, with the preconditions defined section 1.2.1.

1.2.1 Preconditions

The study is done from a company's perspective and is based on the following prerequisites:

- There are only a few developers, three or less.
- There is already an existing web-application that is to be used by the mobile-application.
- The company has a need and/or desire to extend the functionality of it's web-application with native functions.

1.3 Contribution statement

Throughout this work we have been working together very closely. We have reviewed and improved each others work continuously. During development we often been pair programming. With that said we can attribute parts of our work to one of us more then the other. Even though most of the work has been created together.

Philip designed the code of the web application and the mobile application built natively in Android. David designed the code for the PhoneGap application. David researched different techniques for the communication between the PhoneGap layer and the web application layer. David wrote the Approach section and Philip wrote the Introduction section in this paper.

1.4 Report organization

1.5 Terminology

Mobile application
Refers to the application being developed, excluding code loaded from remote websites.

Web application
If nothing else is specified it refers to the client-side of the web application.

Development methods
Refers (in the context of this paper) to the methods of mobile application development methods being evaluated. I.e developing the mobile application with PhoneGap or natively in Android.

Native function
A hardware function which a device has. Lets say a mobile has a camera and an accelerator. When writing software for that mobile there are functions to interact with the camera and accelerator. Those functions are called native functions.

Web/Mobile application layer
Refers to the application layer belonging specifically to the mobile application or the web application. A mobile application can encapsulate a web application which then is a a part of the mobile application. The mobile application then consists of a mobile and web application layer.

1.6 Background

Android is an operating system used on a wide range of devices. For example a mobile device or tv device. Developing Android applications are written in the programming language Java.

A hybrid application is a native application which is partially written with web technologies. I.e HTML5, CSS and JavaScript. The part of the application written with web technologies runs within a browsers engine in a so called WebView. A web application running in a browser does not have access to a device native functions such as the bluetooth. However a website running within a WebView can through interaction with the application's native code access a device native functions.

PhoneGap is a framework for developing mobile applications. The code is written with web technologies. The code can be compiled to different platforms, such as Android or IOs specific code. The resulting application is a hybrid application. An example of a mobile application built in PhoneGap is Wikipedia's mobile application.

1.7 Source lines of code

Source lines of code is a measurement tool for software development. Source lines of code, also abbreviated as SLOC, is very easy to obtain and is a fairly accurate predictor of development effort[1, p. 63]. Measuring SLOC simply means you count the number of lines of code. There are many different ways to measure SLOC, such as Halsted's approach, function points, physical SLOC and Logical SLOC.

Physical SLOC is the length of the code excluding comments and blanks. Function points measure functionality and can therefore be measured before the design and coding if the requirement specification is complete[1, p. 187]. Halstead's uses measurable properties such as operands and operators and uses them to identify properties of software. Such as the length, difficulty and effort of the program. Fenton and Bieman describes Halstead's software science measures as a confused and inadequate measurement. Particularly for other attributes then size[2, p. 345].

Logical SLOC measures the number of statements that carry over one or more physical lines. For languages with terminators, this can be counted more easily and quickly. As an example, in Java you could count the logical SLOC by counting the number of line-terminating semicolons and closing curly brackets. Logical SLOC represents the programming instructions and data declarations which are converted into executable instructions, i.e. the implementation of the software design. Another positive aspect of of logical SLOC is that it better handles differences in formatting and style conventions than physical SLOC[1, p. 155].

To compare size between two different languages a size conversion table can be used. The table can be used to estimate how many SLOC a program coded in one programming language would have in another language. In the table constructed by Galorath and Evans you can compare a third generation language, a fourth generation language, Ada, Assembly or Pascal[1, p. 163]. A third generation language compared to another third generation language would have no conversion rate.

Source lines of code is a measurement tool for software development. Source lines of code, also abbreviated as SLOC, is very easy to obtain and is a fairly accurate predictor of development effort[1, p. 63]. Measuring SLOC simply means you count the number of lines of code. There are many different ways to measure SLOC, such as Halsted's approach, function points, physical SLOC and Logical SLOC.

Physical SLOC is the length of the code excluding comments and blanks. Function points measure functionality and can therefore be measured before the design and coding if the requirement specification is complete[1, p. 187]. Halstead's uses measurable properties such as operands and operators and uses them to identify properties of software. Such as the length, difficulty and effort of the program. Fenton and Bieman describes Halstead's software science measures as a confused and inadequate measurement. Particularly for other attributes then size[2, p. 345].

Logical SLOC measures the number of statements that carry over one or more physical lines. For languages with terminators, this can be counted more easily and quickly. As an example, in Java you could count the logical SLOC by counting the number of line-terminating semicolons and closing curly brackets. Logical SLOC represents the programming instructions and data declarations which are converted into executable instructions, i.e. the implementation of the software design. Another positive aspect of of logical SLOC is that it better handles differences in formatting and style conventions than physical SLOC[1, p. 155].

To compare size between two different languages a size conversion table can be used. The table can be used to estimate how many SLOC a program coded in one programming language would have in another language. In the table constructed by Galorath and Evans you can compare a third generation language, a fourth generation language, Ada, Assembly or Pascal[1, p. 163]. A third generation language compared to another third generation language would have no conversion rate.

1.8 Related work

Chapter 2

Approach

In order to research the problem formulation of this paper, a practical approach has been used. To evaluate and compare the two different methods (as described in problem formulation) a mobile application with the same requirements has been developed in each method. Both of the mobile applications make use of the same web application.

2.1 The mobile applications

The web application that is used composes of a single page and is functional for the web browser Google Chrome. The page displays a form with a text field, an image field and a location field. To get the image or the location value for the form the user can only press a button. When the image or location button is pressed the web application uses the Google Chrome's built in ability to take a picture or get the location.

The mobile applications consists of two parts. The web application described above and the mobile application encapsulating the web application. The mobile application must encapsulate the web application and display the web application exactly the same as it is displayed in a browser. The mobile application must use the mobile's native functions to get the value for the image and the location.

The text field in the web application form must function the same way as it functions in Google Chrome.

For getting the image value in the form the mobile application must give the user the option to use an existing image from the image gallery or take a picture using the camera on the phone. The image that is passed back to the web application must be of in Base64 format.

The location must be obtained using the mobile's GPS.

When the user interacts with the mobile application the interaction must be directly with the web application encapsulated within the mobile application.

In the project the following was used for developing and testing:

- Android SDK version 21
- Apache Cordova CLI version 4.0
- Google Chrome version 44.0.2403.157
- Nexus 5
- Android version: 5.1.1
- Kernel version: 3.4.0-gbebb36b

2.2 Measuring lines of code

The resulting mobile applications are measured using the software metric logical lines of code. The logical lines of code have been counted including and excluding the following:

(here we have a bullet list of what's included and what's excluded)

The mobile application are written in two different programming languages, Java and JavaScript. To compare the measurement result of logical lines of code we use the conversion table described by Galorath and Evans[1, p. 163]. JavaScript and Java is both third generation languages and therefore no conversion is needed.

The logical lines of code written in the web application layer is also measured in both developing methods. To give an correct result we measured the added logical lines of code that has been written as a result of the mobile application. The resulting lines in the mobile and web application layer is added to give a total result.

The architecture of the mobile applications impact the amount of lines of code. The measurement result is therefore analysed using the measurement data and our personal development experience.

2.3 Layer communication

A goal with the mobile application is to have a developer friendly way of communicating between the mobile and web application layer. The web and mobile application has a master and slave relationship. Where the web application layer acts as the master and the mobile application layer acts as the slave. To evaluate this communication a flow diagram of the web layer sending a request for native data and the mobile application layer responding with data is constructed. The diagram acts an suggestion in how developer friendly the communication is and is combined with our personal developing experiences.

Chapter 3

Evaluation

The evaluation is divided into two sections, one for each development method used. In each section, the developed application, it's structure and functionality is presented, followed by the results of the code evaluation.

3.1 Native android

The developed native application can be cloned from Github at [..], also a compressed version of all source code used in the thesis can be found at [..]. An overview of the application structure and functionality follows in section 3.1.1.

3.1.1 Application structure

A UML-diagram representing the Android application can be seen in figure 3.1, with the classes summarized below.

MainActivity is the main class of the program, containing Android-specific code related to startup logic, activities to be started and shutdown of the application. It also serves as a messenger, sending back data from native functions upon completion.

IntentFactory provides intents for native functions

AndroidHardware handles hardware related logic, such as file creation.

Bridge serves as a bridge between the native application and web. When a native function is requested, Bridge requests the corresponding Intent from IntentFactory, and creates a Callback of the corresponding type. The Intent is sent to MainActivity for execution, and the Callback is stored in MainActivity.

WebViewDataSender is the class communicating back to the web application by invoking javascript-functions in the browser.

JsInterface is exposed to the web application as a JavaScriptInterface and acts as a receiver for function calls from the web application. Whenever native functionality is needed, the javascript in the web application can call upon it's methods. JsInterface forwards the function calls to Bridge, which in turn, contains the logic for each function.

Callback contains logic specifying how to send results from a native function back to the web application.

ImageCallback is used when the web application requests an image from the camera.

ImagePickCallback is used when the web application requests an image from the gallery.

UniqueInteger is used by MainActivity when storing callbacks. An unique integer is used as an id for an Intent and its related callback. Upon completion of an Intent, the Callback with the same id is executed.

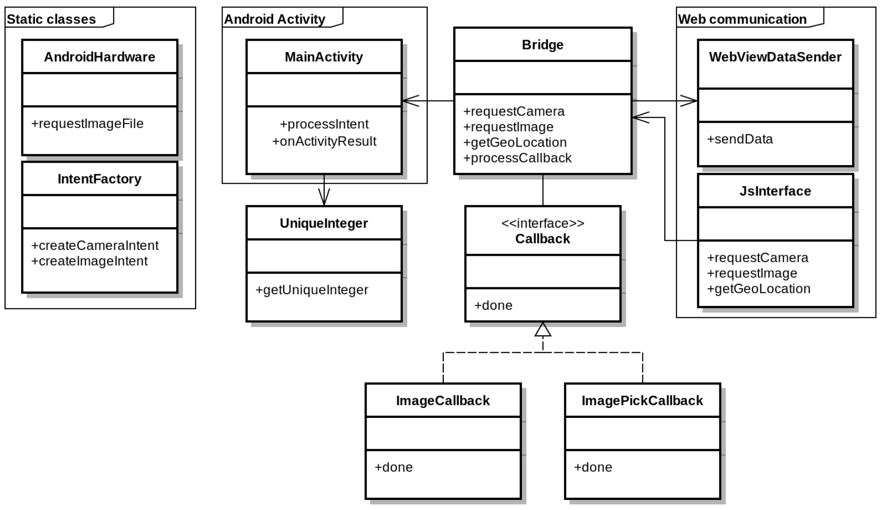

Figure 3.1: Structure of the android application

In order to simplify development of the web application with support for both web browsers and the Android application, part of the logic follows the adapter pattern. The business logic of the web application stays the same, and only the communication with the unit (web browser or application), for example when uploading a picture, differs. Figure 3.2 displays the basic structure of the web application.

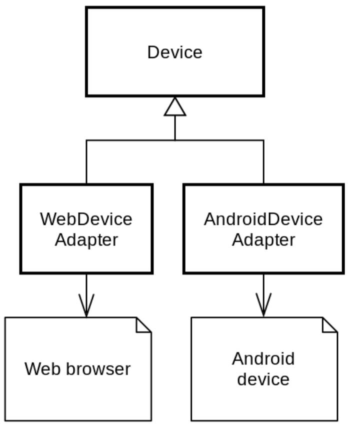

Figure 3.2: Structure of the web application

A more detailed view of the program flow when the web application requests an image can be seen in figure 3.3, further explained below.

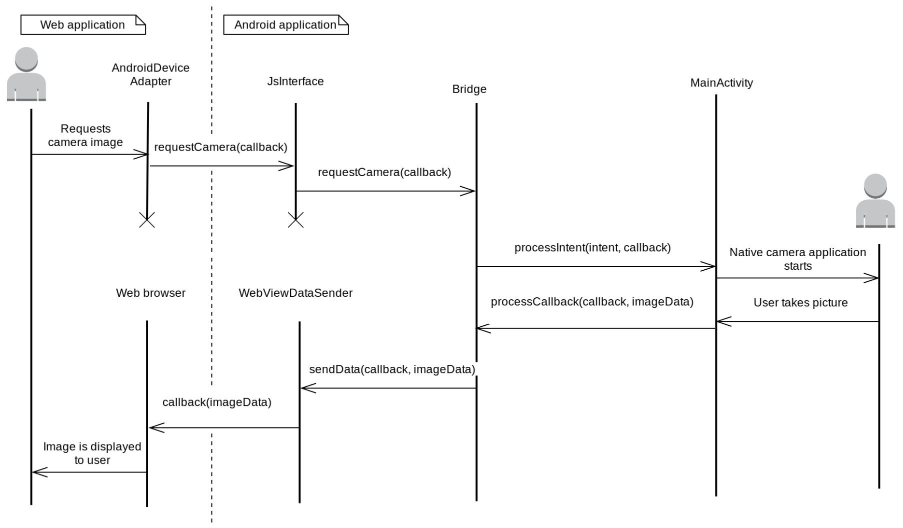

Figure 3.3: Function flowchart, single call from web application

1. User action invokes a request for an image from the mobile camera
2. Current device adapter (AndroidDeviceAdapter) calls requestCamera on JsInterface, with the function name of the preferred callback function as argument
3. JsInterface forwards the function call to Bridge
4. Bridge creates the corresponding Intent and Callback, and calls processIntent on MainActivity with Intent and Callback as arguments
5. MainActivity Creates gets a unique id from UniqueInteger, stores the callback in a hashmap with the id as key, and starts the intent with the id as requestcode.

6. Camera application starts
7. When user takes a picture, MainActivity gets notified by the Android system, pulls the Callback with the same id as the processed Intent, and forwards the callback and resulting image data to Bridge through processCallback
8. Bridge executes the callback which in turn calls sendData on WebViewDataSender, [redacted] with the image data and callback name as arguments
9. WebViewDataSender executes the javascript callback function in the browser window
10. The captured image is displayed to the user in the browser

3.1.2 Code evaluation

Evaluation of the developed application was performed as described in [...], and a summary of the results can be seen in the table below.

Quantitative metrics		
Metric	Android application	Web application
Files	10	2
Logical Lines of Code LLoC	200	173
Single Line Comments	28	19
Multi Line Comments	9	1
High Quality Comments	37	16
Strings	38	66
Numeric Constants	3	9

Result of code evaluation using ProjectCodeMeter

3.2 PhoneGap

The developed PhoneGap application can be cloned from Github at [..], also a compressed version of all source code used in the thesis can be found at [..]. An overview of the application structure and functionality follows in section 3.2.1.

3.2.1 Application structure

A UML-diagram representing the PhoneGap application can be seen in figure 3.4, with the classes summarized below.

Receiver listens to messages posted to the browser window PhoneGap is running in. Receives messages from web application and calls the requested methods in the PhoneGap application.

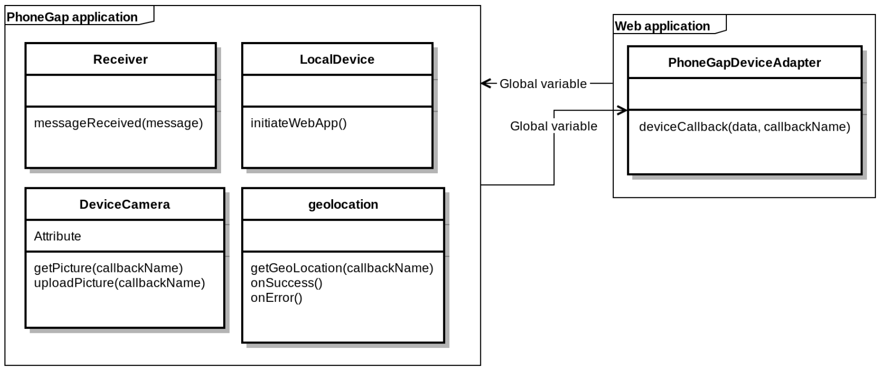

Figure 3.4: Structure of the phonegap application

LocalDevice is used to load the web application and initialize the device in the web application to the correct adapter.

DeviceCamera contains logic for invoking, and receiving results from the device camera. Also forwards the results to the web application.

geolocation contains logic for accessing the phones geo location (GPS-position) and forwarding the results to the web application.

In order to simplify development of the web application with support for both web browsers and the PhoneGap application, part of the logic follows the adapter pattern. The business logic of the web application stays the same, and only the communication with the unit (web browser or application), for example when uploading a picture, differs. Figure 3.5 displays the basic structure of the web application.

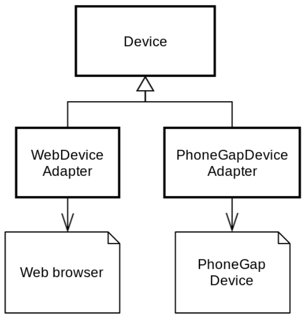

Figure 3.5: Structure of the web application

1. User action invokes a request for an image from the mobile camera
2. Current device adapter (PhoneGapDeviceAdapter) forwards the request to the PhoneGap application using postMessage
3. The message gets processed by the PhoneGap receiver which forwards the call to DeviceCamera via the getPicture function
4. DeviceCamera uses PhoneGap library code to start the camera application

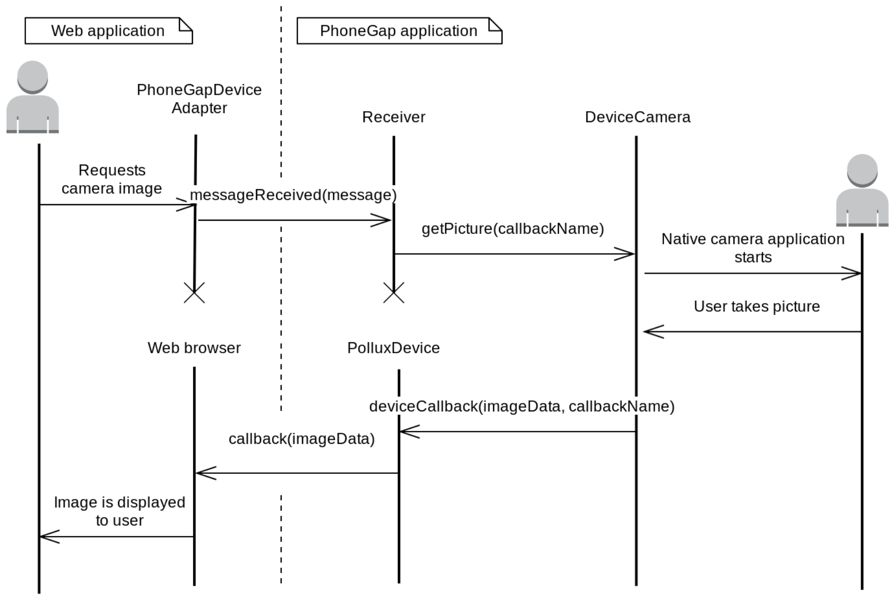

Figure 3.6: Function flowchart, single call in web application

5. User takes picture, and the result is received by DeviceCamera
6. DeviceCamera returns the result to the browser by invoking DeviceCallback on the exposed javascript object, in PhoneGap known as PolluxDevice
7. DeviceCallback calls the javascript callback function specified in its argument
8. The captured image is displayed to the user in the browser

3.2.2 Code evaluation

Evaluation of the developed application was performed as described in [...], and a summary of the results can be seen in the table below.

Quantitative metrics		
Metric	PhoneGap application	Web application
Files	3	2
Logical Lines of Code LLoC	86	175
Single Line Comments	27	19
Multi Line Comments	2	1
High Quality Comments	27	16
Strings	49	68
Numeric Constants	1	9

Result of code evaluation using ProjectCodeMeter

Bibliography

[1] Daniel D. Galorath and Michael W. Evans, *Software Sizing, Estimation, and Risk Management*, Auerbach Publications, Taylor and Francis Group, 2006.

[2] Norman Fenton, Queen Mary University of London, UK, James Bieman, Colorado State University, Fort Collins, USA, *Software Metrics: A Rigorous and Practical Approach*, CRC Press, Taylor and Francis Group, 3rd edition, 2015

[3] Bernard Kohan and Joseph Montanez, *Native vs Hybrid/PhoneGap App Development Comparison*, `http://www.comentum.com/phonegap-vs-native-app-development.html`, Comentum, San Diego, January 26, 201EffectiveUI5

[4] Online survey from Harris Interactive on behalf of EffectiveUI, United States, September 30 - October 4, 2010

[5] Project Code Meter, `www.projectcodemeter.com`, September 8, 2015

[6] Project Code Meter - Logical Lines of Code, `http://www.projectcodemeter.com/cost_estimation/help/GL_lloc.htm`, September 8, 2015