KTH Computer Science and Communication

Building Blocks: Utilizing Component-Based Software Engineering in Developing Cross-Platform Mobile Applications

Master of Science Thesis
Interactive Media Technology at KTH Royal Institute of Technology

By Oskar Andersson (oande@kth.se)

Supervisor at KTH: Vasiliki Tsaknaki
Supervisors at Ericsson Research: Michal Koziuk & Johan Sjöberg

Acknowledgements

I would like to thank Stefan Ålund and Ericsson Research for the opportunity to work with this thesis. In particular, I have to thank my supervisors, Michal Koziuk & Johan Sjöberg, for their help and support during my time there. I would also like to thank my supervisor at KTH, Vasiliki Tsaknaki, for her constructive criticism and valuable feedback.

Thank you!

Oskar Andersson
October 3 2014, Stockholm

Abstract

Contemporary approaches to cross-platform mobile application development, such as hybrid apps from PhoneGap and generated native apps from Xamarin, show promise in reducing development time towards Android, iOS and other platforms. At the same time, studies show that there are various problems associated with these approaches, including suffering user experiences and codebases that are difficult to maintain and test properly.

In this thesis, a novel prototype framework called Building Blocks was developed with the purpose of investigating the feasibility of utilizing component-based software engineering in solving this problem. The prototype was developed towards Android along with a web interface that allowed users to assemble an Android app using software components.

The report concludes that component-based software engineering can be – and already is – utilized successfully to improve cross-platform mobile app development with special regards to user experience. Qualitative data indicate that Building Blocks as a concept is flexible and shows promise for mobile app development in which functionality is often reused, such as enterprise apps. Rapid prototyping using the web-based visual editing tool was another promising area. However, future use of Building Blocks would require further work on the prototype to improve its ease of use.

Table of contents

List of figures

Definitions

Throughout this thesis a number of technical key terms will be used. Here are the definitions that will be used throughout the report.

Cross-platform mobile development	Development of software that aims to target several mobile platforms at once
Platform	A (mobile) operating system, e.g. Android
OS	Operating System
App	Mobile application
Native app	An app developed through native source code, e.g. Java for Android
Hybrid app	A web app wrapped inside a native app, can access some native functionality
Web app	A web site with more functionality, accessed through the browser
Generated native app	An app developed through one language, which is then generated/translated to native source code of other platforms
Building Blocks	A new framework for cross-platform mobile development, proposed in this thesis
WebView	An embedded browser for native apps. Can be used to create hybrid apps
GUI / UI	Graphical User Interface / User Interface
HTML5	A new standard of features for the web. E.g. Canvas, Web Storage, CSS3
WebRTC	Web Real Time Communication, a new web standard for communication between web clients, e.g. using video chat
Black box	When the source code is hidden
CBSE	Component-Based Software Engineering
Jailbreaking	Unlocking the permissions of a mobile operating system, allowing root access to the user
High-level	A degree of abstraction in software. Low-level would be closer to machine code than high-level
Single-page application	A web app where the UI is dynamically generated using asynchronous JavaScript on a single template HTML page

1. Introduction

This chapter is an introduction to the problem that will be investigated.

1.1 Problem background

Although the iPhone 2007 was initially planned to only support web applications for the Safari web browser, native apps are today the norm across all mobile platforms (VisionMobile, 2014 Q3).

With a mobile market split between Android, iOS, Blackberry and Windows Phone in regards to operating systems (IDC 2014, Net Applications 2014), interest for cross-platform development has however kept web technology (HTML5) relevant to developers (VisionMobile, 2014). Since all smartphones have web browsers that (to varying degrees) conform to web standards, web apps may offer the promise of one implementation for all platforms, instead of costly native implementations (Java for Android, Objective-C/Swift for iOS, etc.) for each platform.

The use of HTML5 in app development, has, however been controversial. In 2012 Mark Zuckerberg, (CEO of Facebook) commented that one of the biggest mistakes of their company was “betting too much on HTML5” (Olanoff, 2012). Mobile web app developers Sencha were however quick to respond by creating their own HTML5 version of the Facebook client, called “Fastbook”, which showed that mobile web apps indeed can be performing well (Avins & Nquyen, 2012).

At the same time, a survey consisting of over 7,000 app developers conducted early 2014 point to several problems with web-based apps, such as performance hits, lagging behind native API’s, lack of tools and fragmentation in browser implementations (VisionMobile, 2014 Q1). Recent research in evaluating cross-platform frameworks for mobile apps also point to the same conclusion (Sommer & Krusche, 2013).

Web-based and native mobile apps are generally used in different contexts by developers today. The former typically for applications that need to be quickly developed through well-known web technologies for several platforms, and the latter when development time is less important than delivering the best possible user experience (Sommer & Krusche, 2013).

An interesting question to ask is therefore if it is possible to reach a middle ground, where mobile apps can be developed quickly towards several platforms, but also have a great user experience?

This question has been asked by the academic community in several studies. In 2013’s Automated Software Engineering conference the concept of Cloud Twin was introduced, a way to replay a native Android application on a Windows Phone device over network (Holder, Shah, Davoodi & Tilevich, 2013). A “model-driven” framework called MD2 was introduced in 2013, in essence a new domain-specific language that generated native apps in Android and iOS (Heitkötter, Majchrzak & Kuchen, 2013).

There are also several organizations out there today that try to achieve this. AppGyver's Steroids is a product where the developer can utilize high-performing native UI controls with web content – all with web-like source code to publish to several mobile platforms (a.k.a. a hybrid app). AppGyver also recently released a graphical web tool called Composer to quickly bootstrap a project. It features UI components and an application logic editor (AppGyver, 2014).

Another popular cross-platform development framework is called Appcelerator, which works in a similar way as Appgyver in it using web technology (HTML, JavaScript, CSS) to create hybrid apps. They include a "Marketplace" where developers can find and buy software components to help speed up the development process even further (Appcelerator, n.d.). Native Android and iOS SDK's offer software components such as UI widgets for their app developers (Android Developers, n.d.). They save developers time and offer a way to quickly bootstrap an app. Many software companies such as Facebook offer SDK's in order to help connect third party apps to their services, with, for example, ready-to-use login modules.

Software components and libraries have historically offered tested, optimized solutions to various problems in software development, often related to cross-platform development. In other words, the concept of component-based software engineering already exist in app development today – as UI widgets for native Android/iOS or as commercial Marketplace modules for Appcelerator. What if one was to take CBSE in mobile cross-platform development a step further – to a higher abstraction level? Perhaps one could design a common interface to allow for components to be implemented towards different mobile platforms, but be built and combined in a highly abstracted (high-level) fashion? For instance a modifiable text chat component that would logically work the same on Android and iOS, and then combined with other components to form a working app – towards all these platforms. In accordance with CBSE concepts, these components would be highly tested and optimized for each platform for the best end user experience. Such a scheme could potentially offer speedy cross-platform development while retaining platform specific features. When confronted with the problems of cross-platform mobile app development, these components could provide developers with the building blocks needed to bridge the gap between web and native apps today.

1.2 Purpose

The purpose of this report is to investigate how cross-platform mobile application development can be improved using high-level component-based software engineering.

The first step will be to investigate state of the art technologies for cross-platform mobile application development (e.g. frameworks such as Appcelerator). In particular, focus will be on exploring methods for delivering cross-platform apps that provide a great user experience – that is, a native look and feel that smartphone users today take for granted.

Drawing from the results of the state of the art analysis, a new novel framework called Building Blocks will be proposed and implemented as a proof-of-concept. The goal of Building Blocks will

be to investigate how cross-platform app development can be improved by utilizing high-level CBSE. The framework will then be qualitatively evaluated through interviews.

1.3 Delimitations

The state of the art analysis will focus on the most recent and popular frameworks in use today. This is to filter out old, deprecated and abandoned projects that are no longer relevant for developers today, which helps bring down the study to a manageable amount of frameworks to evaluate. Internet search and statistics will be used to find these frameworks, but not exclusively.

This thesis will focus on one key aspect of cross-platform apps: end user experience. This is because it is generally viewed as a critical factor in mobile apps today (as explained in sections 1.1 and 2).

The Building Blocks framework prototype will only be implemented towards one platform – Android. This is deemed sufficient as a proof-of-concept to showcase the framework, in order to keep the scope of the thesis within a reasonable level. The choice of Android as well as a web-based graphical interface for the framework is explained in more detail in chapter 5.

Since software (app) development is a unique process for each individual doing it, a qualitative evaluation of the Building Blocks framework will be made rather than a quantitative survey. This is to be able to demo the framework in person to each interviewee, and catch unique viewpoints and thoughts on the framework with possible discussion. Interviews will be conducted with researchers and software developers with sufficient knowledge in the domain.

1.4 Problem definition

The leading research question of this thesis is:
"What is the potential for high-level component-based software engineering in cross-platform mobile application development, with a particular focus on end user experience?"

To be able to answer this question, the following subtopics will be explored throughout this report:

- What are the biggest challenges to cross-platform mobile app development today?
- What are the most important factors in successful cross-platform apps?
- How should a cross-platform mobile app development framework that utilizes CBSE – with focus on end user experience – be designed?
- Can a proof-of-concept show if this design is viable or not for real world problems?

The rest of this investigation is aimed at answering these questions.

2. Background

This section covers concepts important to understand cross-platform mobile app development and its challenges today. Web apps and hybrid apps are two key approaches discussed in this section. The concept of component-based software engineering (CBSE) is also introduced and explained.

2.1 Web apps

Web apps are essentially JavaScript applications developed for web browsers with HTML & CSS for UI, usually powered by a backend service. They can be accessed through desktop web browsers, but in this report they will be mentioned only as when accessed through the browser of a mobile device.

When Apple 2007 released the first iPhone, the initial idea was that mobile apps were to be built using web technology for the Safari web browser (Apple, 2007). In fact, Steve Jobs in particular was reluctant to open iOS to third party native apps, which would leave less control in the hands of Apple. However, both internal and external pressure from board members and app developers soon gave way to what later ended up as a mobile app revolution with millions of apps in Apple's App Store, including a new commercial tagline; "There's an app for that!" (9to5Mac, 2011; Ars Technica, 2012)

With today's fragmented smartphone market dominated by Google's Android, Apple's iOS and Microsoft's Windows Phone, the question of web apps is once again highly relevant. As desktop developers historically struggled with cross-platform availability across Windows, Mac OS and Linux, web technologies promised a solution to this problem. Since web standards are adopted by all major web browsers (although to varying degrees), the notion of web apps offers availability in different operating systems and platforms (Sommer & Krusche, 2013). Today we're seeing more and more advanced web applications built with JavaScript & HTML5.

This idea of cross-platform web apps naturally carries over from desktop to mobile. So why would you hire several developers to write native code for all different mobile platforms, if you could get away with fewer developers writing web apps? Why weren't the first iPhone's web apps a success?

First of all, web applications have to abide by the rules of the browser it lives in. This means that hardware features such as camera, accelerometer data, etc. are subject to each browser implementation, and is typically a lot less open because of security concerns (VisionMobile, 2014; Sommer & Krusche, 2013). Secondly, the kind of performance, UI-wise, that native applications give the user is difficult to emulate in a web app. End users tend to notice immediately when, for instance, a button animates a second too late (VisionMobile, 2014; Sommer & Krusche, 2013). Facebook's switch from HTML5 (web technology) to a native application reportedly gained them great performance gains (Dann, 2012; Du, 2012), further cementing the view of web apps being too slow.

Notable speaker Paul Irish from Chrome Devtools had a presentation in 2013 where he talked about how the power of desktop machines today have lulled developers into making web apps that simply aren't optimized enough for the memory and power constrained environments of smartphones (Irish, 2013). According to Irish, mobile web browsers (especially native WebViews) are typically 6-10 slower than desktop counterparts due to the hardware (and software) differences (Irish, 2013). This means that mobile web applications – powered by said mobile browsers – need to be built in a more resource-conservative way, with a meticulously optimized performance in terms of JavaScript, DOM-manipulation and CSS.

Because web technology (JavaScript in particular) is so flexible and open, it's easy to make mistakes that the more structured programming languages of Java or Objective-C may have forced the user to code in another, possibly more efficient way, where the developer can leverage UI components for Android and iOS built by Google and Apple (Hemel & Visser, 2011). Even if the web/hybrid app is greatly optimized, the end user will often experience an "Uncanny Valley" when comparing web UI to native UI, due to small oddities creating a feeling that something just isn't right (Fowler, 2011). The team behind Google's web framework AngularJS had a presentation in late 2013 where they explained how keeping a user waiting more than one second in an app causes them to think the app is broken (AngularJS, 2013).

Academic research points to the same conclusion regarding performance. In a recent master's thesis (Friberg, 2014), the author found that cross-platform applications (web and hybrid) had an inferior performance when compared to native apps, and would go as far as to limit their usefulness to less complex apps. Other recent academic articles point to the same conclusion (Heitkötter, Majchrzak & Kuchen, 2013 and Sommer & Krusche, 2013). However, Friberg (2014) pointed to opportunities in the future as frameworks keep progressing. In fact, mobile web advocates such as Sencha's CEO Michael Mullany is hard at work trying to convince people that HTML5 actually is mature enough, with good performance (Mullany, 2013). It may be viewed as self-serving work for Sencha as a web framework company, but "Fastbook" is a tangible example of their work, and can be tested with a mobile device at http://fb.html5isready.com/.

At the same time, the late 2013 presentation by AngularJS mentioned above as well as Paul Irish explained how mobile web app developers should simply think differently to provide a better user experience. According to these sources, it is very much possible to create an app with a good user experience if the app is built using a different mindset than traditional desktop web apps; utilizing techniques such as CSS3 transforms, splash screens, more advanced debugging and profiling tools, AngularJS's "ngTouch" module for touch events and more (AngularJS 2013, Irish 2013).

A 2013 article presenting a web-based digital publishing framework for mobile web apps called "Stage" also identified possibilities for improved performance with CSS3 transforms, and pointed to the importance of different web browser implementations in the performance of web apps (Aamulehto, Kuhna, Tarvainen & Oittinen, 2013), suggesting that web apps indeed can be a viable alternative to native implementations.

2.2 Hybrid apps

In a highly cited article titled “Mobile application development: web vs. native” (Charland & Leroux, 2011), a future is presented where projects such as Apache Cordova help to access mobile native functionality – e.g. GPS and accelerometer data – to create a good mix between web and native – the hybrid app. In a way, this is an evolution from the pure web apps of 2007 living in the browser, as these hybrid apps are created as a native app, wrapped around a WebView element. The native WebView element acts as an embedded browser that contains a web app, but the wrapping native code allows the web application to access additional native features such as camera, accelerometer, etc.

Apache Cordova has implementations of these native features in each platform (Java for Android, Objective-C for iOS) that are exposed through a public JavaScript API in each WebView for these platforms (Apache Cordova API Documentation, n.d.). Figure 1 shows an example of this, presenting how accelerometer data can be accessed from JavaScript code.

```
function onSuccess(acceleration) {
    alert('Acceleration X: ' + acceleration.x + '\n' +
          'Acceleration Y: ' + acceleration.y + '\n' +
          'Acceleration Z: ' + acceleration.z + '\n' +
          'Timestamp: '      + acceleration.timestamp + '\n');
};

function onError() {
    alert('onError!');
};

navigator.accelerometer.getCurrentAcceleration(onSuccess, onError);
```

Figure 1. A JavaScript example for accessing a native accelerometer in a hybrid app (Apache Cordova API Documentation, n.d.).

Today, however, hybrid applications are still plagued by some of the same problems as in 2011 and earlier, most of which are related to the implementation of WebViews. These may be performance issues, lack of tools or differing (or complete lack of) implementations of new browser features such as HTML5 standards or WebRTC (VisionMobile, 2014; Traeg, 2013). Google’s recent announcement of an improved WebView (called ChromeView) for Android 4.4 may indicate that things are improving in this regard, though (Google Chrome, n.d.).

It’s also the opinion among developers that Cordova and other frameworks will simply lag behind native feature support. As a new native API is released, like a temperature sensor for Android, someone needs to actually develop this plugin for Cordova. Often, the plugin may be developed for only one platform.

Quote from a mobile software development manager in a recent developer survey (VisionMobile, 2014 Q3): *“The gap between HTML and native is widening. The native SDKs introduce new*

APIs at a faster rate than HTML5 can keep up with. To harness the power of these new APIs native is the only viable option."

However, with new projects such as AppGyver's Steroids and Monaca strongly driving the progress of cross-platform frameworks, new techniques may lead to a new generation of hybrid apps which offer a level of user experience approaching that of native apps. The new approach that they have proposed allows creating applications built of several WebViews bound together with native UI controls. Basically, the web code is used here to displaying content, while a special XML syntax is utilized to call for native UI features (Traeg, 2013). Figure 2 presents an example of this new type of hybrid app. Of course, this dependency on native UI features (which must be implemented by AppGyver or Monaca) limits the customizability of the UI, but may provide a better user experience.

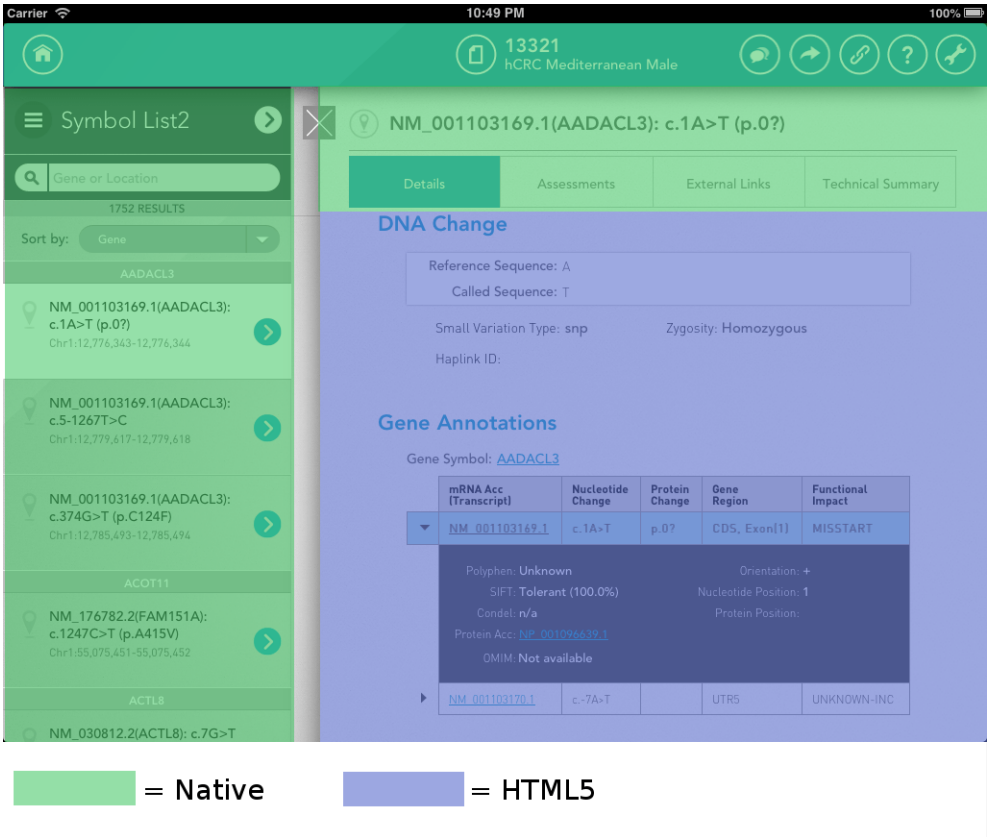

Figure 2. Showcases what technologies could be used where in a hybrid app (Traeg, 2013).

2.3 Cross-platform mobile app development

A wide array of cross-platform development frameworks have emerged in recent years, such as Appcelerator Titanium, PhoneGap, MoSync and others that simplify and streamline the development of hybrid and native apps based on HTML5/JS and other techniques (Wikipedia, 2014a). In the case of generated native apps, these frameworks often utilize a sort of "semi-web" (XML with special annotation/syntax) to produce native UIs, such as Titanium or MoSync

(MoSync, n.d.), or require specifically named JavaScript functions to interact with native functionality and might look completely different from their web counterpart.

Basically, these frameworks try to create an abstraction layer spanning several different languages used on different mobile platforms, which may be problematic because languages solve things differently. The abstraction may not cover all special cases, and will be limited to the lowest common denominator of all languages involved in terms of functionality (Wikipedia, 2013c). If one were to switch framework, reusing old code would also be more difficult as the code base only works for one framework. Because of this, choosing a cross-platform framework often results in a degree of vendor lock-in due to the unique code base of that framework. Other problems include debugging and testing code intended for several platforms, as well as an inherent inability to conform to the user interface guidelines of each platform (Wikipedia, 2013c) because of a level of abstraction that removes the specific choices each platform offers.

Many of these frameworks utilize components or plugins in a way. These components may be corner stones of their architecture (such as MoSync and PhoneGap's dependency on plugins to access native functionality). The components may also be part of the general app development ecosystem, such as Appcelerator's Marketplace, where developers using the framework may buy or sell their own components. In part because of this, the concept of component-based software engineering will now be introduced and discussed. For an in-depth analysis of relevant frameworks, see chapter 4.

2.4 Component-based software engineering

Component-based software engineering (CBSE) is a widespread concept that has been around since the early 90's. The idea is that by encapsulating specific functions into a software module that is thoroughly optimized, tested and documented, reusability will be of great value to a developer (Aoyama 1998, Wikipedia 2014b). Today, hybrid mobile development components at large make use of low-level Cordova plugins that access native functionality, such as the camera or accelerometer. A scientific paper aimed at summarizing CBSE research (Wulf, Pipek & Wun, 2008), states that *"Component technologies are perceived as an important means to keep software architectures flexible."* As such they can be found today in all types of software ranging from backend to mobile frontends, though the ways for implementation of the concept vary greatly.

During Google I/O 2014 in June, Eric Bidelman from the Google project Polymer gave a presentation where the concept of "thinking in components" was strongly advocated. He talked about introducing a new web standard called "Web Components" – a way for web developers to abstract their web code into smaller, logical parts, and allow for code reuse by finding available components. Bidelman argued that web development today lacks common structure with different development APIs for every framework, which results in inefficient code that is hard to survey. The Web Components web standard could provide a solution to this problem according to Bidelman. Figure 3 shows a Web Component search engine that helps developers to find the component they need.

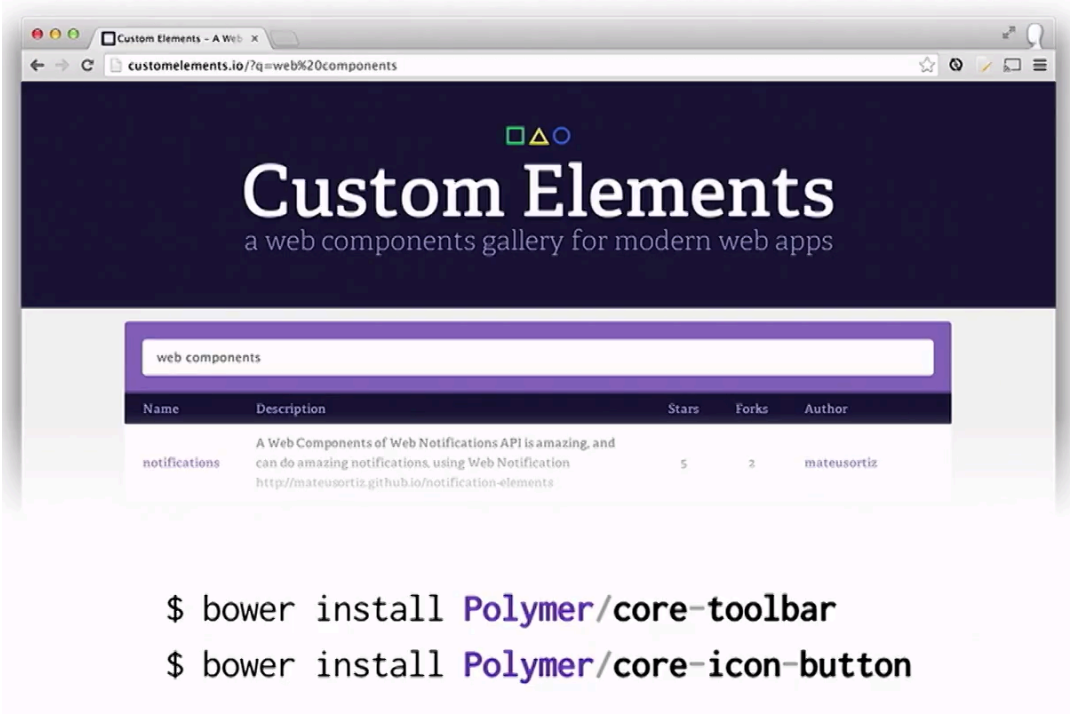

Figure 3. A Web Component search engine (Polymer, 2014).

The cross-platform mobile application development framework Appcelerator has since 2011 had a “Marketplace” for shopping for “Modules” (a sort of mobile app component) (Claburn 2011). In this Marketplace, one can find video editing components, barcode scanners, etc. to readily use in new app development projects (Appcelerator n.d.).

One of the most cited books on the subject on CBSE, Component-Based Software Engineering: Putting the Pieces Together (2001) offers an extensive explanation of the concept and compares it to traditional engineering to a large extent. An example mentioned is within construction, where you build bridges using ready-made components, such as steel pillars. The gain of utilizing CBSE is in not having to reinvent the wheel all over again, and to be able to break up a big problem into smaller, reusable parts – much like the gain coming from object oriented programming (OOP). According to the authors, component developers (producers) are responsible for testing and optimizing their component, which other developers then leverage when using those components in an application.

The difference to OOP is that CBSE needs to provide means for encapsulation, or even black-box (hidden) source code, as component developers might not want to release their source code. Each component should follow a specific Component Model that dictates a structure or interface that all components must follow, meaning that an application should be able to be constructed by simply assembling different components (Heineman T. & Councill W. 2001, p. 37). This composition can be made programmatically or visually. If one were to visualize this component composition – thus raising the abstraction level – a major challenge is the lack of structure in graphical representation, according to the authors (p. 43).

A problem identified is one of documentation in CBSE, bringing up the example of the Ariane Rocket disaster where the lack of proper documentation of software components resulted in a disastrous rocket launch. Seeing as implementations can be hidden or difficult for vendors to understand (as they did not write the code themselves), proper documentation is of critical importance (Heineman T. & Councill W. 2001, p. 244).

In a study from 2008, CBSE research, and in particular visual component composition was investigated. Their findings in part support Heineman & Councill's statement that visual component composition is challenging and that documentation is critical, as they realized the need for strong documentation in order for test subject (in the case of the study, completely inexperienced programmers) to understand each component in a visual component composition tool. See picture 3 for a screenshot of a part of the tool evaluated.

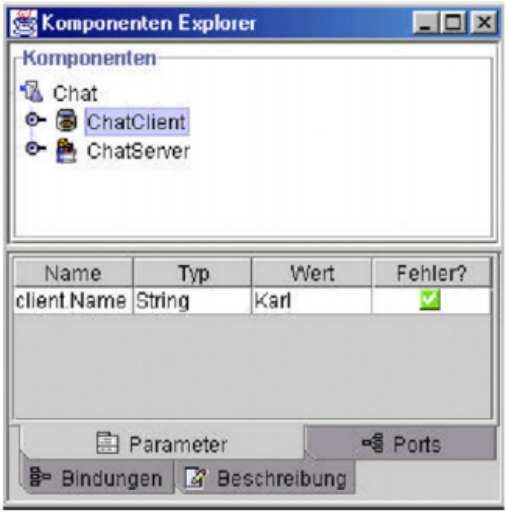

Figure 4. A screenshot from the Component Explorer of the 2008 CBSE and visual component composition study (Wulf, Pipek and Wun, 2008).

3. Methodology

Several methods were used to carry out the research work. First of all, a literature study provided a valuable general outlook on the problem area. Then, a state of the art (SOTA) analysis was carried out to specifically identify challenges to the problem today. Based on results from the SOTA analysis, a prototype framework was developed and evaluated using interviews.

3.1 Literature study

Google Scholar as well as KTH Royal Institute of Technology's "Primo" service have been used extensively to search for academic work. These search engines have been also supplemented by ordinary Google Search, which has been used to find newspaper articles, Wikipedia pages and other relevant sources.

Keywords used when searching included: "cross-platform", "mobile", "web", "html5", "hybrid", "application", "component-based software engineering", "component-based mobile application development" as well as others. In order to get relevant material for this relatively new problem domain of mobile apps, results have been filtered to the years 2012-2014, although concepts such as "cross-platform" and "component-based software engineering" have had no restrictions in this regard since these concepts are not as new.

Articles were also found in the journal *Automated Software Engineering (ASE)*, the conference journal *Proceedings of the 21st ACM international conference on Multimedia*, *International Journal of Human-Computer Studies*, *Web Information Systems and Technologies* as well as *Communications of the ACM*.

3.2 State of the art analysis

A state of the art (SOTA) analysis was carried out in order to identify the challenges to the problem area of cross-platform mobile app development.

In the SOTA analysis, Google Search, Google Scholar and the KTH Primo service were used in order to find both academic and industry solutions and discussions of the problem area. A list of popular frameworks for developing cross-platform mobile apps was put together using Wikipedia (2014a) rankings as well as academic and industry comparisons and evaluations.

The frameworks from the list were then analyzed and investigated further by looking at internal API documentation, code examples and external articles, where problems and opportunities could be discovered.

3.3 Prototype framework

The results of the SOTA analysis were used to design a new novel framework for creating cross-platform mobile apps. The development process of implementing the framework prototype

was iterative, where findings uncovered during development were continuously used to improve the design. The final implementation of the framework prototype has then been used as a proof-of-concept for the novel approach proposed in this thesis.

An important reason for developing a prototype was to evaluate and study a proof-of-concept of a novel solution to the problem at hand. As high-level component-based software engineering – the core concept of the proposed solution – is a concept that is hard to evaluate quantitatively, it was critical to be able to carry out user testing.

3.4 Qualitative evaluation of the prototype

The framework would lastly be evaluated qualitatively through interviews consisting of four experienced software developers and researchers who possessed sufficient domain knowledge of the problem area. The prototype was demonstrated to the interviewees who could also try it out and look at the source code. The goal was to gather a wide range of ideas and feedback on how and if the framework could be beneficial in real life situations.

The interviewees were asked about 17 questions each, and the questions can be found in the appendix of this report. For the sake of anonymity, the interviewees were named A, B, C and D.

4. State of the art analysis

The aim of this section is to present different kinds of approaches to cross-platform mobile app development, and to analyze problems and opportunities in each approach.

A list of enterprise products and university-sponsored research projects was investigated. The list was put together using two criteria:

- Novelty (unique technology)
- Relevance (latest release not older than three years)

The projects are grouped together in seven categories according to their architecture and the kind of apps they generate.

4.1 Enterprise products

4.1.1 Traditional hybrid apps: Apache Cordova (PhoneGap) / MoSync / RhoMobile

PhoneGap, MoSync and RhoMobile are all frameworks that provide tools for developers to create hybrid apps. PhoneGap is built on top of Apache Cordova. These tools offer possibilities to compile and test apps, through cloud services or downloadable software. These hybrid apps utilize the scheme described in chapter 2.2 of exposing native functionality to JavaScript, but depend largely upon HTML/CSS for the UI (MoSync, n.d.; Wikipedia, 2014d; Wikipedia, 2014a). These frameworks either use open source Cordova plugins for this native functionality, or provide it themselves.

In a conference paper titled “Evaluating Cross-Platform Development Approaches for Mobile Applications” by Heitkötter, Hanschke & Majchrzakand (2013) from the University of Münster, Germany, the choice of PhoneGap was identified as a viable choice only *“if very close resemblance to native look & feel can be neglected”*. Other reports point to the performance problems with hybrid technology, where the implementation of the native WebView is a major bottleneck (Aamulehto, Kuhna, Tarvainen & Oittinen, 2013).

Surveys (VisionMobile, 2014 Q3) also show that simply keeping up with new native API changes is a big challenge, essentially needing a new Cordova plugin each time. Another problem that occurs is that the code base gets cluttered. Basically, the developer ends up with an app that is part web app, part custom native functionality calls. This means that the app is harder to test and maintain.

4.1.2 New generation of hybrid apps: Monaca / AppGyver Steroids

Monaca and AppGyver Steroids are two new initiatives (from 2013) that expand on the traditional hybrid app architecture. In short, they promote separation of web code by allowing for several WebViews, rather than enforcing a single-page application (see Definitions) that may be slow to load for resource-constrained mobile devices. Screen transitions and UI controls are primarily left to native UI components (Tanaka, 2013; Lehtimaki 2013).

This solution in part takes care of the performance issues of hybrid apps by introducing native UI components, but reinforces the hybrid app problem of code clutter. The source code of the app is now a mix of a web app – which can be debugged and tested in a browser – and custom native feature interfaces (XML files or JavaScript/Coffeescript calls), such as for screen transitions, which cannot be debugged in a browser.

And yet performance in the remaining WebViews is still a problem, which means that in order to maximize the use of Monaca or Steroids, one should avoid adding any user interface to this part of the app. Effectively, this means that the app in its layout and design is severely limited.

4.1.3 JavaScript-generated native apps: Appcelerator Titanium

Appcelerator Titanium takes yet another step away from hybrid apps in abandoning WebViews altogether. In fact, it's purely based on a native UI. The architecture is similar to using Cordova plugins, but to a much larger extent. The only remaining web code is JavaScript, and has its own XML for layout (which will be converted to native UI) and its own TSS for styling (similar to CSS for the web). JavaScript is used for the business logic of the app and is powered by Appcelerator's own JavaScript engine that is bundled in each app.

This architecture shows promise because the user experience is much more fluent and fast with all native components. It may however struggle to keep up with native APIs, or fail to enable developers to use platform-specific UI because of its development abstraction that aims to work with all mobile platforms.

However, due to the ability to deploy on several platforms at once with a good user experience, Appcelerator has become one of the most popular cross-platform development frameworks, and with a Marketplace for selling and buying software components such as video chat, development is even faster (Wikipedia, 2014e, Appcelerator n.d.).

4.1.4 Other generated native apps: Xamarin / MoSync / Qt

Xamarin, MoSync and Qt all share this grouping because they generate native apps based on a single language, similar to Appcelerator Titanium, but without any JavaScript at all. Although MoSync is listed for hybrid apps, they also offer a second approach similar to Qt in writing apps in C++ through the use of "translation libraries" of C++ code to Java or Objective-C, which compiles to native source code. Xamarin does more or less the same thing in C#, but bundles the .NET runtime with apps. All these frameworks offer XML to edit views, much the same as (for instance) Android, and expose native APIs.

While the apps generated by these frameworks may be on par with native apps in terms of performance, a challenge for these frameworks is to keep up to date with official APIs of Android, iOS, etc. Another problem is a limited user experience that settles for the lowest common denominator, as the user interface must work for all platforms.

Among these frameworks, Xamarin stands out as one of the most popular ones for cross-platform mobile app development. A high rated post in the software development community StackOverflow by Luigi Saggese (2013) gives a few compelling reasons for choosing Xamarin over PhoneGap and Appcelerator Titanium:

- Native UI performance
- One codebase (C#) = testability
- IDE with Android and iOS visual design
- Component store (Xamarin, n.d.) for faster development

Products	Dev lang	Compiles to	Architecture
Apache Cordova / MoSync / RhoMobile	Web	Hybrid app with one WebView	Uses platform-implemented plugins to expose native functionality to JS
Monaca / AppGyver Steroids	Web (with Cordova plugins)	Hybrid app with (optionally) several WebViews	Based on Cordova. UI XML files (.ui) / JS / coffee for native layout . Can use native controls / transitions, JS logic
Appcelerator Titanium	JavaScript, XML	Hybrid app with native UI, using a JS engine	Alloy: XML (≈HTML), TSS (≈CSS), JS based on backbone.js. Classic: all in JS
Xamarin / MoSync / Qt	C# / C++	Generated native app	.NET runtime bundled or C++ compiled to native source code, native APIs available, XML for views

Figure 5. Cross-platform app development products.

4.2 Research projects

4.2.1 Cloud Twin

Cloud Twin is a 2013 research project from Virginia Tech, USA, which aims to solve the problem of cross-platform mobile app development with a network-based approach (Holder, Shah, Davoodi, & Tilevich, 2013).

The general idea is simply to replicate and replay an app from one platform to the other. It works by translating a source XML (UI) tree to the target platform, while replicating UI calls over (cloud) network. In the prototype developed, one can replicate an Android app to a Windows Phone app.

Specifically, Cloud Twin uses web sockets to transmit UI actions performed on the Windows Phone app to a server, where the actions are played out on an Android emulator, which then replays the correct UI response back to the Windows Phone app.

While the authors believe that Cloud Twin could be a viable option in this market, they admit that network delay and customizability of its UI (which depends on the XML tree) limits its

usefulness. In fact, they state that its range of applicability is very limited due to its dependency on the UI XML tree, which cannot easily be modified to work with, for example, animations.

4.2.2 Mobl / Stage Framework

Mobl (Hemel & Visser, 2011) and Stage Framework (Aamulehto, Kuhna, Tarvainen & Oittinen, 2013) are two other research projects (the former for a PhD, the latter for a MSc) whose approaches lead to web apps designed for mobile.

Both projects implemented a sort of domain-specific language in order to more easily develop these apps; Mobl's was entirely new, while Stage depended on HTML and JSON. Mobl cited a need to abstract away "boilerplate" web code, while Stage cited a need to easily produce web apps optimized for mobile.

All the issues and opportunities with web apps listed in chapter 2.1 still apply, however, and won't be covered here.

4.2.3 MD 2

MD 2 is a PhD research project from 2013 that also introduces a new domain-specific language, which in contrast to Mobl or Stage, generates a native app (Heitkötter, Majchrzak & Kuchen, 2013). The authors cite a need for better native performance than today's hybrid solutions can offer. The prototype developed can compile MD 2 source code to Android and iOS native code and can access native hardware such as the camera, but will like Xamarin and other frameworks inevitably lag behind new native API updates. Another issue is also the fact that none of the code in this framework can be transported for any other use, which is not the case for web and hybrid apps, effectively vendor-locking developers in this framework.

According to the authors, the targeted group for this app would be enterprise applications, where customizability of UI is not paramount. The types of apps generated through this framework are deliberately limited in range of applicability to fit this data-driven enterprise app outlook.

Research	Dev lang	Compiles to	Architecture
Mobl (2011 PhD)	Domain-specific	Web app	New language to abstract away “boilerplate” web code. Uses an MV-pattern
Cloud Twin (2013 research paper)	Java	Generated native Windows Phone (WP) app	A scheme for replicating an Android app to a WP app. A source XML (UI) tree is translated to the target platform, while replicating UI calls over (cloud) network
Stage Framework (2013 MSc)	HTML, JSON	Web app (magazines)	Its JS, HTML & CSS3 structure makes up a magazine viewing web framework
Md2 (2013 PhD)	Domain-specific	Generated native app	New model-driven language. MVC-pattern of “entitites”, “views”, “actions”

Figure 6. Cross-platform app development research projects.

5. Proposed framework: Building Blocks

The theoretical idea behind the Building Blocks framework is introduced here. An architectural overview of the implemented proof-of-concept is also presented, with reasoning behind design choices.

5.1 Guidelines

The literature study and state of the art analysis revealed that existing frameworks for cross-platform mobile app development all had various issues. The choice was made to create a new framework that attempted to solve the problem with a novel approach. Five particularly important factors in successfully developing cross-platform mobile apps were identified and used as guidelines for the new, proposed framework:

1) **The importance of native UI for the best end user experience.** This is one of the biggest reasons why mobile web apps have failed to take off.
2) **Accessibility of native functionality, such as GPS and camera, is critical.** Support for native functionality is the main feature of hybrid apps.
3) **Components are used in various forms to successfully speed up the development process.** As illustrated by, for example, Appcelerator Marketplace and Xamarin Components.
4) **Customizability of UI and functionality to fit each platform; avoiding the problem of lowest-common denominator.** One of the biggest problems of generated native apps.
5) **Maintainability and testability is critical.** In general a very important factor in software development that is lacking in hybrid approaches.

5.2 Proposed framework architecture

The main idea of the Building Blocks framework is to enable the use of a new type of cross-platform components, here called blocks, with the purpose of improving cross-platform app development according to the identified guidelines (section 5.1). These blocks can have UI or simply be utility classes, and are implemented in each platform's native language. Because any possible UI or hardware access is implemented in the target platform language (e.g. Java for Android), native UI (1) and native functionality (2) from the above guidelines are achieved.

By following this common interface (in CBSE terms a "Component Model"), blocks will be built to allow for customization and reuse by different developers in different scenarios. The main goal, however, is that these blocks should be implemented towards several platforms according to the same common interface.

Thanks to the common interface, these blocks can then talk to each other and be combined to form an entirely new application. This combination step can either be abstracted to a new high-level domain-specific language or a visual editor. Because the new application utilizes components that are implemented for several platforms, the application itself can be generated to these platforms as well.

Testing, optimizing and maintaining the app is done primarily in each separate block (by its producer) in accordance with CBSE principles. Developers leveraging these blocks consequently save a lot of time and effort. Because the resulting app is generated to each target platform consisting of blocks written for that platform, testing can also be done as if it was a native app, thus satisfying (5).

Overall, (3) should be satisfied when there are enough blocks in the framework so that app developers have to build a minimal amount themselves, and can focus mostly on composing and modifying different blocks. The only question is whether or not to allow blocks to be black box, effectively limiting (4). Open source blocks would mean that the layout (and logic) of the app could be customized to the best detail possible in each mobile platform.

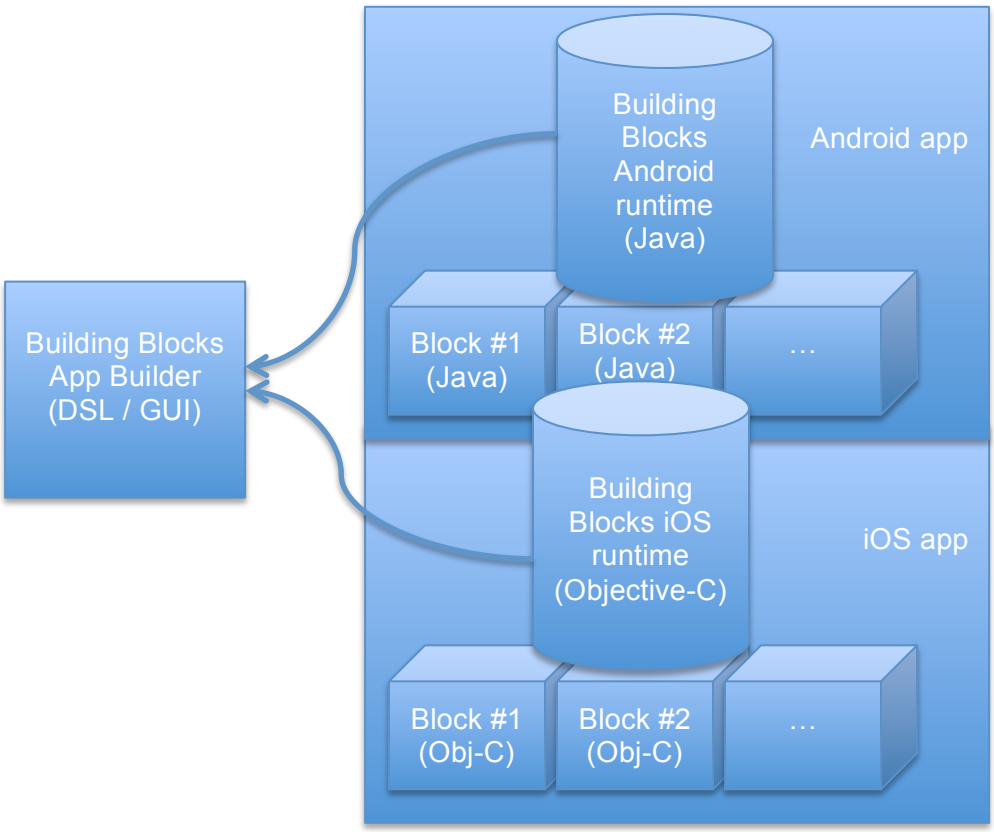

Figure 7. A theoretical overview of Building Blocks with two platforms.

5.3 Prototype architecture

As a proof-of-concept, an Android prototype was developed. Android was chosen due to its openness as an open source platform, ease of use and prior experience. A desktop web GUI was also developed in order to showcase the composition process of the blocks. The web was

chosen in order to be able to quickly deploy a prototype, and because of the ease in creating custom UI. While a new DSL could have been created, it was decided that a graphical tool would be both easier to use by end users as well as to implement.

The Building Blocks prototype consists of three major parts: the App Builder (web GUI), the Building Blocks Android runtime and five sample Android blocks. An app composition file (JSON) contains the output of the App Builder tool (the logic of the Building Blocks app), and a block descriptions file (JSON) describes all Android blocks with available methods and events for the web GUI.

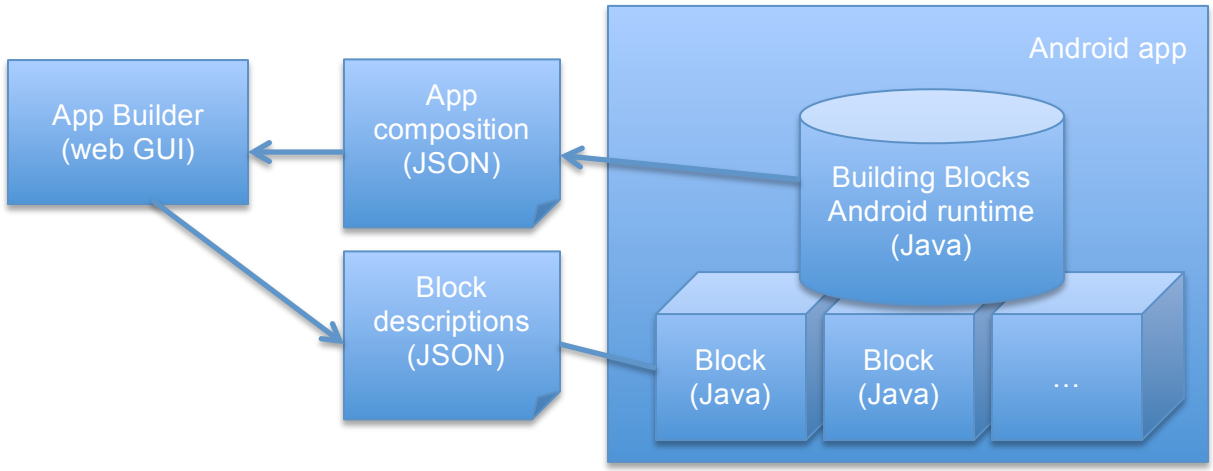

Figure 8. A simplified overview of the Building Blocks architecture.

5.3.1 Building Blocks App Builder (web GUI)

The App Builder is a graphical tool for developers to combine different blocks into a mobile application. Composing and customizing an app in this GUI results in an app composition JSON file (see figure 8). The web GUI was built using AngularJS (for structuring the JavaScript business logic), and Bootstrap (for styling the HTML).

The GUI consists of three views:

- "Add building blocks": search for and add blocks to the app (figure 9).
- "Customize events": customize each block and set up the logic of the resulting app. The logic consists of **events** and **methods** (figure 10).
- "Building complete": overview of the resulting app JSON source code (figure 14).

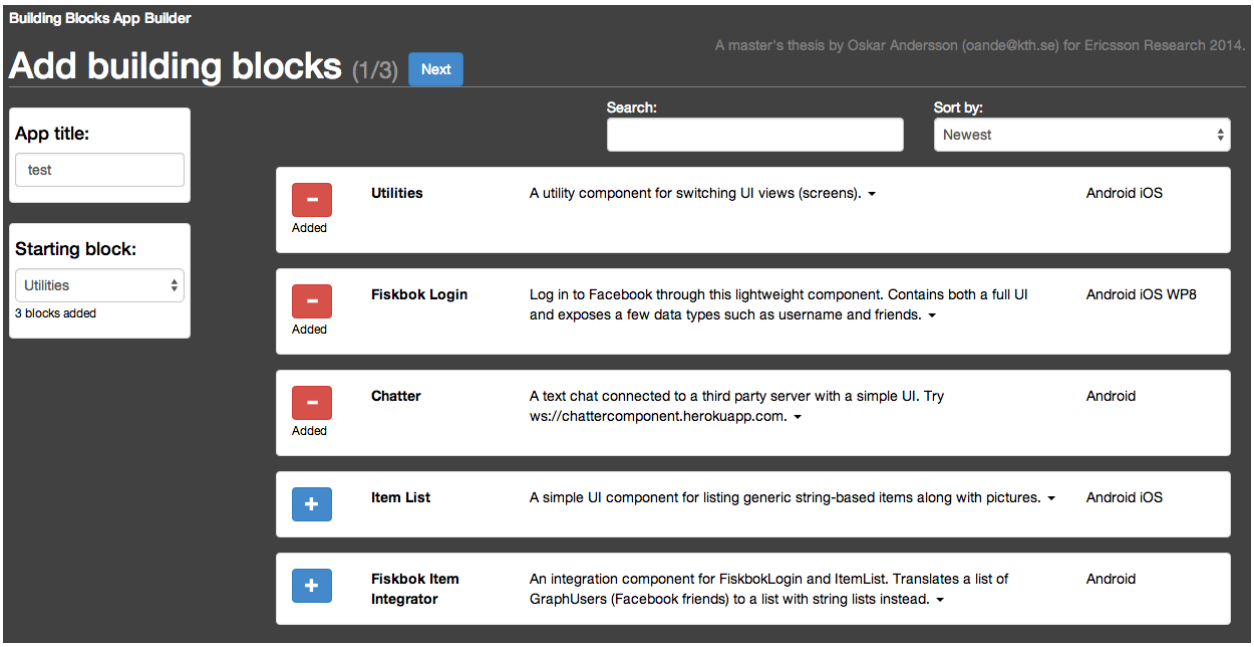

Figure 9. Initial view where one can search for and add blocks to the app.

In the first "Add building blocks" view (figure 9) one can search for blocks by block names, method names, event names and block descriptions. It's possible to click to expand each block and see available methods and events. Three features are purely aesthetic; the app title (left side), starting block (left side) and platforms listed (right side). These features have no function, but give a hint of how it could work in the future.

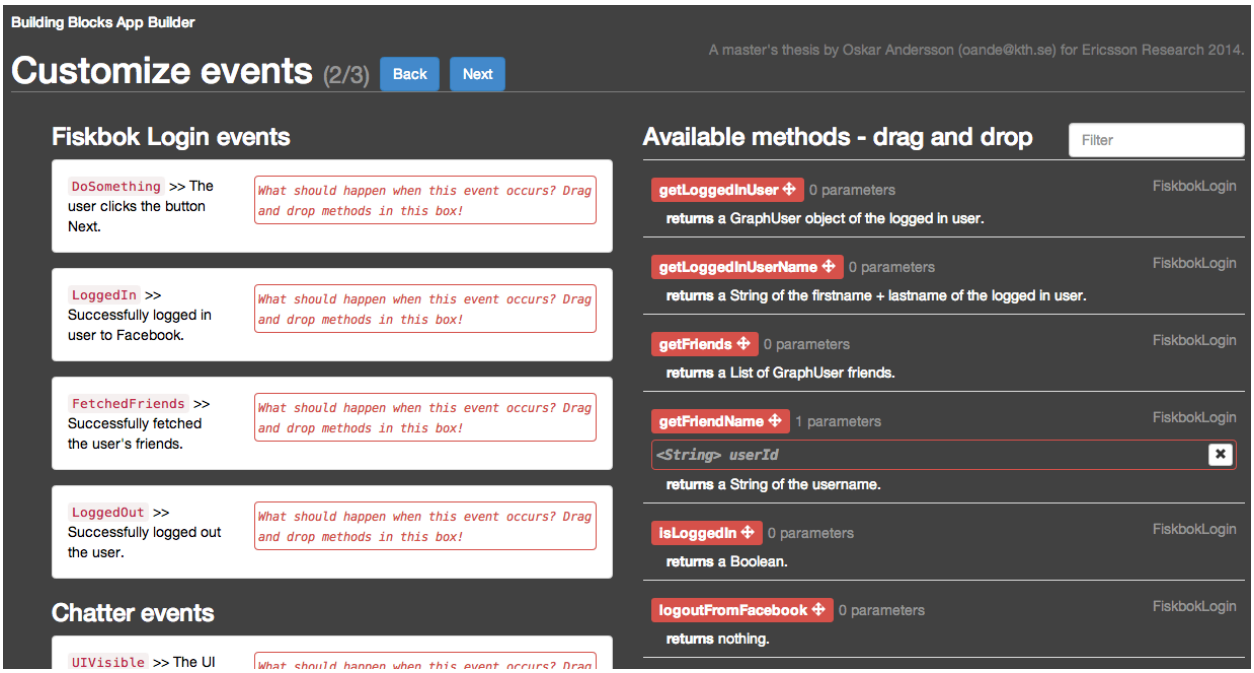

Figure 10. Second view. Here one can customize the events and methods of each block.

Each block contains **events**, which are triggered in various places of an app; for example when a user clicks a button, or when a login attempt to a backend service has succeeded. These events are listed on the left side of the second view (figure 10), and are explained in detail with figures 11-12.

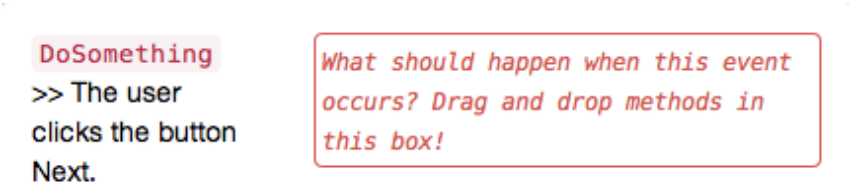

Figure 11. Available event with no registered method.

Figure 12. Event with a method called "startNewComponentScreen" registered to it.

Each block also contains public **methods** with optional parameters. An example of a method can be seen in figure 13. These methods can be dragged in the UI to an event (see figures 11-12), or as a parameter to another method.

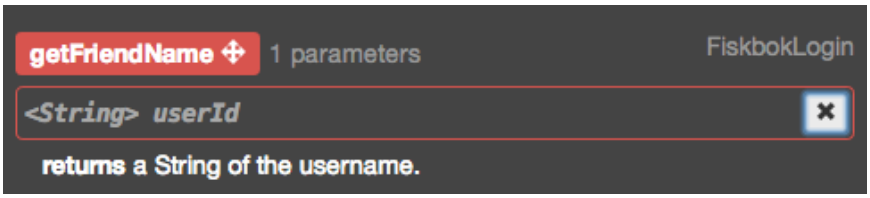

Figure 13. A method called "getFriendName" with a string parameter "userId". "FiskbokLogin" is the block from which this method originates.

After the second view of customizing **events** and **methods**, a JSON source code representation of the new app is created at the third and last view (figure 14). This JSON code is then used in the Android runtime along with the needed blocks to run the new app as a coherent Android application.

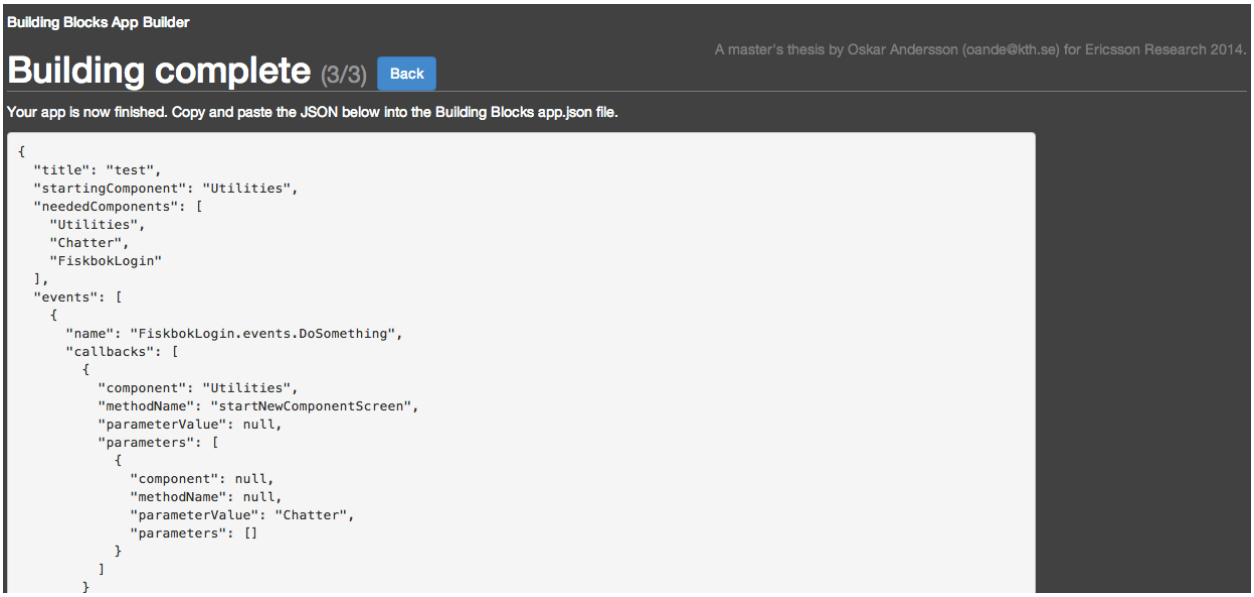

Figure 14. Third and last view. The source code (JSON) of the app composition is shown.

5.3.2 Android blocks in Building Blocks

An Android block is at the bare minimum one ordinary Java class (called a model class in the framework) that exposes public methods. These public methods control the business logic of the block itself, and have to be precisely described in a JSON file to work with the web GUI. The blocks follow a simple Java interface to keep a unique id and starting Activity (if one is provided). See figure 16 for an example class diagram of a block.

This base model class can be complemented with Activities or helper classes. Activities can get the reference to the model class through the Binder by using:
`Binder.getModel(<Activity> thisActivity)`

The blocks can trigger events in the Binder by using:
`Binder.callEvent(<String> newEventName)`

Events can be triggered when a user clicks a button, or when some data is received by a backend service. As shown in figures 11-12, events can be hooked up with methods that are called once the event is triggered.

These blocks are then run as Activities (if they have one) in the Android app, with the Binder class providing the “glue” between these blocks (see figure 15).

5.3.3 Building Blocks Android runtime

The Building Blocks Android runtime is the largest and most complex part of the framework in terms of code. Because of this, only the most important concepts will be covered here. The

purpose of the runtime is to provide a common interface for different blocks to be able to communicate and to act like a coherent Android application.

It solves this in five steps:

1. A static BinderBlueprint class parses the app composition JSON file (which describes the block-based app) to an AppBlueprint Java class. This way all the needed blocks are known.
2. The BinderBlueprint initializes each block's base model class and puts the references in a static list.
3. It populates an event list with a new type of object called ComponentMethod. ComponentMethods are recursively built with direct references to the initialized (from step 2) base model class methods. Once called, they invoke the appropriate block method with any type of parameters.
4. A static Binder class exposes the method to trigger an event from each block. When an event is triggered, the event list (from step 3) is searched and all ComponentMethods hooked to that particular event (e.g. "user clicked button") are called.
5. The Binder and BinderBlueprint classes are implemented in a fail-fast manner by crashing the app or throwing BinderExceptions when incorrect app compositions are parsed in the BinderBlueprint.

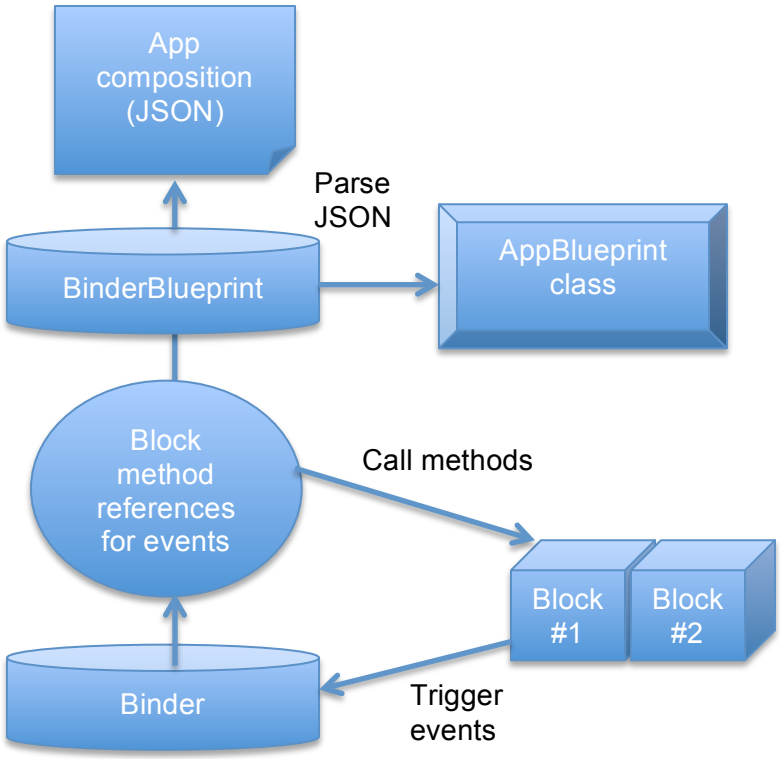

Figure 15. A simplified overview of the Building Blocks Android runtime.

5.3.4 Android sample blocks

In order to be able to have any meaningful demonstration with the prototype, five sample Android blocks were developed (all in Java for Android, except the Chatter server which was written in NodeJS):

- Utilities: a utility block with a helper method for switching Activities, `startNewComponentScreen(<String> newComponentId)`
- Fiskbok Login: a Facebook login service with a UI and some helper methods, such as: `getUserName()`
- Item List: a UI for showing a generic list of Items.
- Fiskbok Item Integrator: an integration block for converting a list of Facebook friends from the Fiskbok Login block to a list of Items that matches the Item List block.
- Chatter: a text chat with complete UI, that connects to a third party server through `connectToServer(<String> address, <String> startingUsername)`

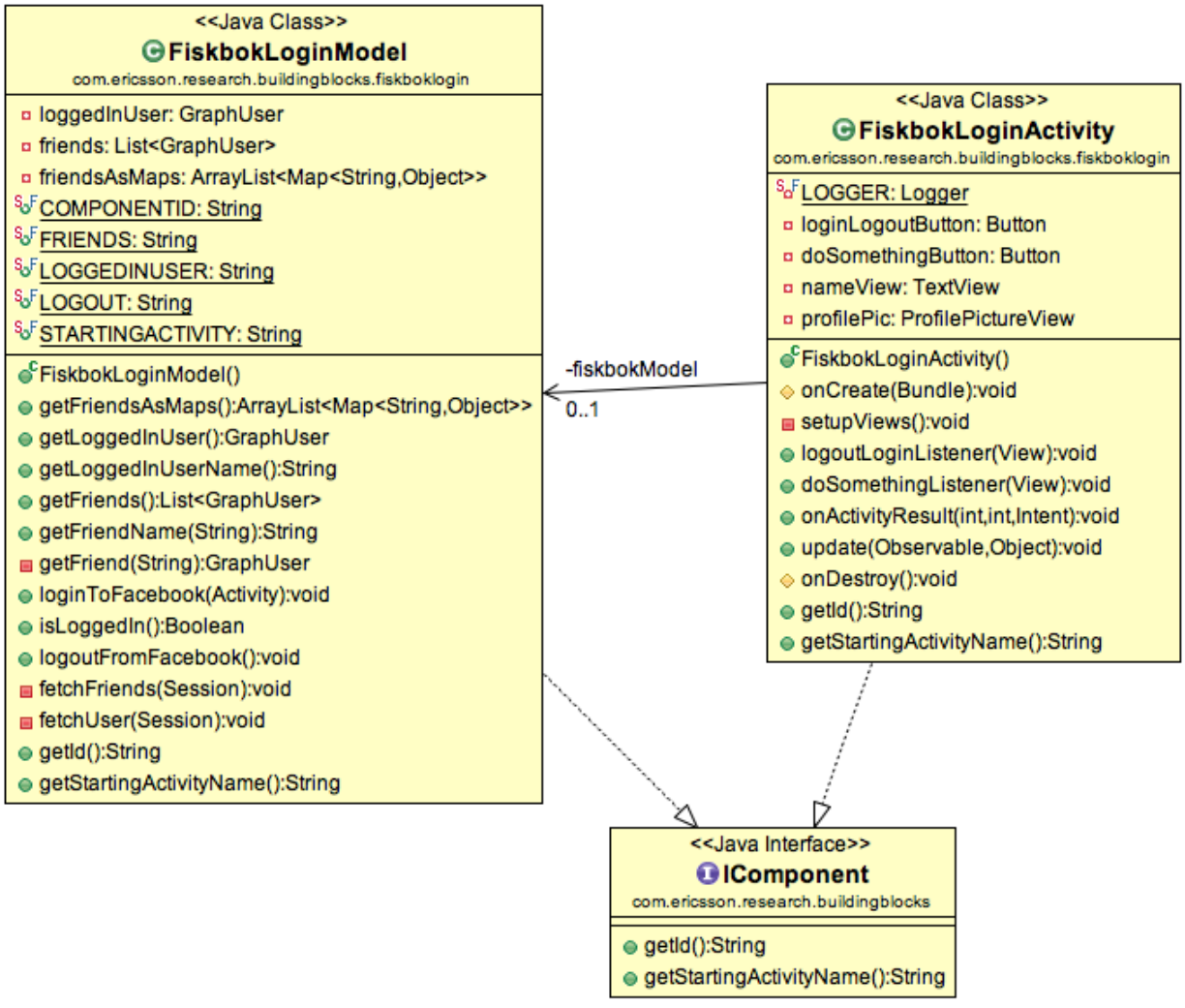

Figure 16. A class diagram of the Fiskbok Login block. FiskbokLoginModel is the base model class with public methods. The fiskbokModel object reference is retrieved from the (not seen here) Binder class.

5.3.5 Compiling

To compile an app, one has to copy and paste the app composition JSON (from the web GUI) to a specific *app.json* file in a ready Eclipse project that contains all blocks, and edit the *AndroidManifest.xml* to set the correct main (block) Activity. Given time, a feature to automatically generate all project files should be trivial to implement.

6. Results

The implemented Building Blocks framework prototype is compared to other frameworks, and the interview responses from the prototype evaluation are presented in this section.

6.1 Building Blocks

Among the five guidelines identified in chapter 5.1, all of them are satisfied in one way or another in the Building Blocks framework Android prototype.

End user experience, platform-specific customization and native functionality

Because the implementation of each component (block) of a Building Blocks prototype app is done natively in Android, native UI and Android-specific features are all supported, satisfying (1) and (4) of the guidelines. This is in contrast to both hybrid apps (such as PhoneGap or Monaca), where native performance is a struggle due to web technologies, and in generated native apps (like Xamarin or Appcelerator), where performance may be good, but native specifics are often missed.

When developing a block, native functionality (2) is therefore also guaranteed with the latest Android SDK available. However, assembling an app from existing blocks means that one has to wait for block producers to implement or upgrade to new Android APIs. This situation of lagging behind official API changes is similar compared to hybrid and generated native approaches.

Component reuse

Of course the biggest gain from utilizing Building Blocks is the possibility for great component (block) reuse across several mobile platforms. Several other cross-platform frameworks have this possibility as well, especially web and hybrid apps, which can reuse existing web libraries and modules. But by introducing an App Builder web GUI for assembling and customizing different blocks into one coherent app, bootstrapping a new cross-platform app is even faster. However, this system depends heavily on whether there are enough blocks available for developers to actually save time. If only 10% of the intended app is available through blocks, perhaps no time is saved at all due to having to write 90% block implementations towards all intended platforms. A critical difference is that all other frameworks provide an easy way to write code *without* components, but everything in a Building Blocks app needs to be built with components.

This is why an effort was made in making sure that creating a component (block) is as hassle-free as possible. In order to build a block for the Building Blocks Android prototype framework, one practically develops an Android application with two key methods. One is for accessing the business logic (model) from a possible Activity, `Binder.getModel()`, the other is for exposing customizable events, `Binder.callEvent()`. The model part of the block is mandatory and contains all public methods that can be used. Two JSON-files are also required – one for

translating block IDs to their main Java classes, and one for describing public methods and events for the web GUI. In contrast to Xamarin Components, Appcelerator Marketplace or even Apache Cordova plugins, Building Blocks components (blocks) are specifically intended to be easily assembled into one coherent app. Another key difference is that the blocks of this new framework are created much more like small, encapsulated Android apps. This means that the threshold of implementing a block is much lower than for instance developing a Cordova plugin, which is implemented in a very specific manner towards a WebView.

Maintainability and testability

In accordance with CBSE principles, testing and optimizing should be done in each component (block) of Building Blocks. Because each block is implemented natively in each mobile platform, mature testing tools for IDE's such as Eclipse/Android Studio or Xcode help satisfy (5) of the guidelines for this new framework. An app assembled from these blocks is therefore supposed to have test coverage for its separate parts, practically unit tests. Of course, this all depends on how well blocks are implemented by their producers. Another problem is that no automated integration tests validate that an assembled app actually works, but since the resulting Android app is an Android project, such tests should be trivial to implement in that mature environment. At the same time, the fail-fast approach of the implemented prototype means that manually testing a compiled app with an erroneous app composition JSON file will crash quickly, in part solving this issue.

Missing features

Currently the framework lacks a feature to automatically pick block source files and generate all the Eclipse project files needed for a project. This is done manually in the current version, including copying and pasting the app composition JSON from the App Builder. Although requiring considerable time and effort, this problem was deemed of trivial research value and could be faked for the sake of framework demonstrations.

6.2 Qualitative study

The qualitative study was carried out with face-to-face interviews with four professional researchers and software developers, here named A, B, C and D to preserve anonymity. It was believed that these interviewees had sufficient domain knowledge of mobile app development to provide valuable insights about the proposed novel framework.

A demonstration of the framework was carried out for each interviewee. The demonstration showed the process of assembling an app from sample blocks in the web GUI. After getting the app composition JSON from the web GUI, it was then used in a ready Eclipse project containing all sample blocks and compiled to a Nexus 7 tablet. After demonstrating that the app worked according to the web GUI app composition, the process for developing an Android block in Eclipse was explained.

After the prototype demonstration, about 17 questions/topics (see appendix) were asked and discussed. The questions ranged from practical aspects of the App Builder web GUI to more topical discussions regarding cross-platform app development and component-based software engineering. The answers represent only the personal opinions of the interviewees.

Personal backgrounds

Interviewees D and A had about 9 years of software development experience, C about 6 years, and B about 2 years. All except D had experience of mobile app development, with primarily Android for B, Android, iOS and Symbian for A and finally iOS, Java ME and web for C. D had mostly experience from programming media frameworks and embedded hardware. All interviewees were around 25 – 35 years old.

The App Builder web GUI

A, B and D thought it was easy to use the App Builder to customize and combine blocks to create an app. C was of the opinion that it was not entirely clear, but emphasized that having once learnt how to use it would save a lot of time in developing applications that shared similar functionality. D believed that the web GUI would be easy for a software engineer to understand and use, but ease-of-use would differ greatly between individuals who were not programmers.

While discussing documentation in the web GUI, B and D believed that it was sufficient for software engineers, but needed more work for other target groups. C requested better explanations of how to combine and modify methods in the customization view of the web GUI, and suggested an additional view where the app could be seen as a web representation. Interviewee A argued that descriptions of methods and input parameters were needed, and to be able to search for components by functionality.

When asked if it would be possible to create any type of app imaginable through the App Builder, B and C expressed doubt and said that much depended on the quality of the components. However, C also said that it was hard to say before the framework had been used more, but that specifically games would be difficult to create successfully. A's and D's answers were more positive by arguing that any app would be possible to create if component source codes were freely available for editing. However, some thought it would be difficult (D) or even unfeasible (B) to create very large/complex apps that required many components, as there was no real overview. D stated that it would be hard to see the connection between screens, and that a redesign of the web GUI might be needed to be able to oversee this aspect properly.

The Building Blocks Android framework prototype

The question of how to create a component (block) revealed that although D thought it was easy, A and B both thought it needed improvement. Interviewees A and B argued that there was too much copying and pasting as of now, and that there could be a tool or better guidelines for generating base classes and skeleton classes with required help methods. In regards to logic customization, all four said it was enough with events and public methods. However, A, B and C

said that it would benefit from being able to send data along with events, and D said that accessing properties without get- and set methods would be an improvement.

Although the demonstration used copying and pasting in a ready Eclipse project, the question of generating project files or compiling directly to an app from the web GUI was brought up. Three (B, C and A) thought it was much better to primarily have access to generated project files for either quick prototyping (B) or simply more options before compiling manually (A, C). D, as someone not involved or experienced in app development, thought it would be great to be able to directly compile to an app for quick testing to be able to focus on other software areas.

There were various responses regarding a question of where the layout and design of the app should take place - in the web GUI or in the IDE (Eclipse). While A thought that customization could take place in both the GUI and the IDE (“Higher-level customization in the web GUI, and lower level in the IDE”), B thought that it would be problematic in terms of synchronization to allow for two ways to design the app. C firmly believed that the IDE was the best option because it would be very difficult to keep up with the mature tools available in Xcode and Eclipse. D argued that this question depended on who used the tool and how it was used; for some enterprises it might be beneficial to provide a way to follow enterprise UI guidelines through static component methods aimed for the web GUI, and that overall he saw no reason to having to use the IDE, that one also should be able to design different platform-specific UIs in the web GUI.

A question about how the interviewees would like to provide documentation if they were to develop a component, revealed that all four would like this to be automatically generated through the source code, similar to how Java Docs works today.

When asked about integration between components, A said that he considered it an easy objective. D thought that it would be easy if the APIs and descriptions were clear enough, but that being able to look into the source code of each component would be ideal. B and C both thought it would be good to have a system that could automatically provide help to convert between data types.

General concepts

Discussions were had regarding the future of components for the framework. A and B believed it would be difficult to sell components, either due to digital rights management issues (B) or simply because licensing would be difficult for single components (A). Instead, C believed in an open source library, and that developers would create their own components instead of buying them if they couldn’t find it. Interviewees A and D thought that there could be a commercial market, either to buy components with a unique value like a backend service (A), although unlikely, or through some specific enterprise platform (D). Interviewee A also expressed possibility to sell a license for the Building Blocks framework for enterprise app development.

While discussing the views on cross-platform development today, B and D stated that they believe that the web will take over this field in the future as web browsers and devices become

good enough. B was critical of frameworks such as Xamarin where one in the end really need to focus on design and user experience in each mobile platform. However, both B and C believed that user experience was not the primary objective for e.g. enterprises, where functionality and low cost is the primary concern. C argued that mobile platforms move so fast, that it's a big problem in getting stuck with the lowest common denominator when using frameworks, because the different mobile platform APIs can't be translated perfectly to all platforms. However, A identified that the concept of cross-platform development overall is very important to consider in order to save development time and money.

Final opinions on the Building Blocks framework

Interviewee A thought that the framework was very flexible, and enough work in the GUI along with access to component source codes could enable building any type of app – given that all components needed were available. Keeping component producers responsible to keep their components updated to match platform versions may however be problematic according to A, especially if their source code is unavailable to change for someone who wants to use it.

Lack of components is something that B identified as a possible problem for the framework. C said that "The more specific the intended app is, the less you gain by using the framework", and overall most interviewees came to the conclusion that the Building Blocks framework shows most promise for app developers who can reuse functionality. Enterprise apps especially would fit in this category, according to C and A, where components can be reused for new purposes or clients.

C also thought that the option for modifying things specifically in each mobile platform was good. However, C also stated that it could be valuable to be able to create functionally simple apps that didn't necessarily have anything to do with cross-platform development. The App Builder could in fact also be used in educational purposes to learn programming.

With regards to usage within a corporate development environment, there were varied opinions. Interviewee A and D thought that the Building Blocks framework could be beneficial in general for building mobile apps, and most (A, B and D) believed it would be suitable for rapid research prototyping. D especially emphasized faster development times and a lower threshold to start programming - something C also believed.

7. Result analysis

In this section, the methodology and results of the prototype implementation and qualitative study will be discussed. Primary differences to other frameworks are summarized.

7.1 Method discussion

Framework toll on user experience

Although the increased screen size of a Nexus 7 tablet compared to a regular smartphone was considered beneficial in showcasing the resulting Building Blocks app, the added hardware performance might have skewed results in terms of end user experience. At the same time, the implemented sample apps were all simplistic and resulted in identical performance on a Samsung Galaxy S4 device. A more diverse (in terms of complexity; performance requirements and hardware demand) set of sample blocks would have been interesting to evaluate on a varied set of hardware devices.

Interviews

In order to get more objective feedback, more interviews would be beneficial. It might also have been good to follow up the interviews with a large survey to probe the industry for feedback as well. This would have given an indication as to whether the opinions expressed in the interviews were isolated or shared amongst the developer community in general. However, in order to answer the research questions of this thesis, the chosen approach should be deemed sufficient, as the outlook was not to prove statistically that Building Blocks is a success or not, but rather investigate and explore a novel approach to a complex problem.

7.2 Result discussion

In this section, some key observations from the qualitative study and the prototype development are presented and discussed.

Shows promise in speeding up enterprise cross-platform app development

One of the key reasons for developing apps using cross-platform mobile app techniques is to speed up the development process. While web, hybrid and generated native apps may have other faults, they succeed in this aspect. One of the most critical points about the Building Blocks framework that the qualitative study and the prototype development (see chapter 5) revealed, is that it relies heavily on what components are available. Just in order to demonstrate the prototype, five sample components were developed. At the same time, the same qualitative study showed that there is a belief that with more polishing of the App Builder, better documentation, and generated project files, the framework could be used directly for enterprise app development, where many apps with similar functionality may be developed quickly. By streamlining the component creation process, and especially the missing project file generation feature described in section 5.3.5, the interview responses indicate that cross-platform development with Building Blocks shows promise.

Documentation and generated project files – the importance of open source blocks

As documentation was a critical aspect of component-based software engineering according to Heineman & Councill (2001), this was investigated during the interviews. It was realized that being able to look at the source code of blocks is very important to understand the functionality, and that simply prohibiting black box components would mean that developers would gain valuable assistance and documentation. Although this may be in conflict with the CBSE principle of allowing component producers flexibility, the interviewees were at best doubtful that a commercial market for Building Blocks components would be viable in the same way as for example Xamarins Components or Appcelerator Marketplace. This may be because components is a much more important concept in the Building Blocks framework. And so generating project files that closely resemble a regular Android application in Building Blocks not only allows for flexible platform-specific UI and features, but also increases the readability and documentation for a project.

Visual composition using the App Builder – for experienced programmers

In contrast to the study conducted by Wulf, Pipek and Wun (2008), the interviewees in the evaluation of the Building Blocks App Builder (the web GUI) were all experienced programmers. This was a deliberate choice as visual component composition is a rather limited field that by design is a speedy process. From a software engineering standpoint, it seems unlikely to be able to build an entire application or the exact component that you need using only what's already available. At one point, one should have to build an entirely new component to fill a gap, therefore needing programming experience. In particular, Heineman & Councill (2001) argued that visual component composition is difficult to realize in a real project, but in contrast to this, three out of four interviewees in this thesis stated that it was easy to understand and leverage in the web GUI. However, as one interviewee pointed out, it may be much harder for individuals without programming experience. This would seem likely, drawing from the conclusions from a 2008 study (Wulf, Pipek and Wun, 2008).

Testing and maintaining for each platform - dependency on tested components

A problem identified in this year's VisonMobile's surveys (2014 Q1, 2014 Q3) is maintainability and testing issues for frameworks that consist of many different techniques and code bases. In large, this is a problem that Building Blocks solves with its component-based, fail-fast approach (described in chapter 5.3). By depending on component producers for providing thorough test coverage for their components (in accordance with CBSE principles), an assembled Building Blocks app should have full unit test coverage.

Although a problem identified in chapter 5 in lacking automated integration tests to validate an assembled Building Blocks app, manually implementing integration tests could easily be done in each mobile platform IDE by developers. For instance, system integration tests could be created with mature tools in Eclipse. A problem, however, is that this requires knowledge of testing in all mobile platforms, which is fundamentally a problem that cross-platform development seeks to solve. It might therefore be a good idea to investigate how the Building Blocks framework itself can provide automated integration testing in the future.

7.3 Differences to existing approaches

There are three key features in Building Blocks that differ from existing frameworks.

Component-based app structure

In Building Blocks, apps are structurally made up of components (blocks) only. These blocks follow the API of the framework so that they can be used together. This is in contrast to PhoneGap hybrid apps that contain mostly custom JavaScript/HTML in conjunction with Apache Cordova plugins for accessing native hardware features. In generated native apps from for example Appcelerator Titanium or Xamarin, component stores do exist, but do not follow a common interface to allow for direct component-to-component communication. As a result, contemporary cross-platform development uses mostly custom code that is harder to reuse, but also easier to work with for specialized projects where it may be difficult to find existing components.

Implemented natively towards each platform

Each block in Building Blocks is implemented in each platform, with access to native UI and features. Generated native app frameworks such as Xamarin also offer this, but may be restricted in using the lowest common denominator for possible platform specific functions. Hybrid apps from PhoneGap, Monaca and others are all limited by WebView implementations that often result in sluggish UI.

Visual app composition

Although most existing frameworks IDE's offer GUI editing, the Building Blocks App Builder takes this a step further in visually editing the application logic. Recently this year, AppGyver released a similarly themed tool called Composer to quickly bootstrap their hybrid apps (AppGyver, 2014). The main difference between the App Builder and Composer is that the App Builder is all about assembling and customizing components, while Composer lets you visualize common programming concepts such as if-conditions to create new functionality from scratch.

8. Conclusion

8.1 Answer to the research question

In the beginning of this report, the following research question was asked:
"What is the potential for high-level component-based software engineering in cross-platform mobile application development, with a particular focus on end user experience?"

The literature study, state of the art analysis, as well as the overall development of the prototype has helped to answer that CBSE can be – and is – utilized successfully to improve cross-platform mobile app development with regards to user experience. Examples such as Apache Cordova's plugins, web libraries such as AngularJS and Web Components as well as generated native app frameworks such as Xamarin's Components all point to this.

Interviews and a proof-of-concept implementation of the Building Blocks framework reveal that it's too early to say whether this new novel concept could be an improvement to cross-platform app development. Instead, the qualitative study of the framework shows that it is flexible, and has pointed to new opportunities in visual application composition such as rapid prototyping or in developing functionally similar enterprise applications – opportunities that were not imagined when this project was initiated.

8.2 Future research

In order to properly evaluate the impact of this new novel framework, it would be of interest to see it used in a series of app development projects. This would require some further development of the framework to enable automatic project files generation and streamlining component creation. This should showcase limitations and opportunities of the framework on a more hands-on, practical level towards solving cross-platform development issues.

The unforeseen opportunities that resulted from the Building Blocks framework – primarily visual application composition for rapid prototyping – would also be interesting to take a closer look at to see how it can be improved or utilized.

References

9to5Mac (2011) *Jobs' original vision for the iPhone: No third-party native apps,* 9to5Mac 21 October 2011, viewed 18 August 2014, from http://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/

Aamulehto R., Kuhna M., Tarvainen J., & Oittinen P. (2013) "Stage Framework - An HTML5 and CSS3 Framework for Digital Publishing". *MM '13 Proceedings of the 21st ACM international conference on Multimedia*, p. 851-854

Android Developers (n.d.) *App Components*, viewed 25 April 2014, from http://developer.android.com/guide/components/index.html

AngularJS (2013) *Angular on the go: Using Angular to power Mobile Apps*, viewed 11 April 2014, from https://www.youtube.com/watch?v=xOAG7Ab_Oz0&list=TL31yG3z-7_y32zCR2wUz5uyLUgmtd6GqM

Aoyama, M. (1998) "New age of software development: How component-based software engineering changes the way of software development". 1998 International Workshop on CBSE.

Apache Cordova API Documentation (n.d.) *Accelerometer*, viewed Aug 11 2014, from http://cordova.apache.org/docs/en/2.5.0/cordova_accelerometer_accelerometer.md.html

Appcelerator (n.d.) *Open Mobile Marketplace*, viewed 10 April 2014, from https://marketplace.appcelerator.com/home

AppGyver (2014) *Composer Is the Fastest Way to Bootstrap High-quality Mobile Apps,* viewed 10 Aug 2014, from http://www.appgyver.com/composer?utm_source=AppGyver+Monthly&utm_campaign=a943dcad20-Composer_Beta_LaunchSteroids5_22_2014&utm_medium=email&utm_term=0_c20355596b-a943dcad20-93945725

Apple (2007) *iPhone to Support Third-Party Web 2.0 Applications*, viewed Aug 18 2014, from https://www.apple.com/pr/library/2007/06/11iPhone-to-Support-Third-Party-Web-2-0-Applications.html

Ars Technica (2012) *"Who needs an app store?" Five years of iPhone*, viewed 18 August 2014, from http://arstechnica.com/apple/2012/06/who-needs-an-app-store-five-years-of-iphone/

Avins J. & Nguyen J. (2012) *The Making of Fastbook: An HTML5 Love Story*, Sencha Blog 17 December, viewed 2 July 2014, from http://www.sencha.com/blog/the-making-of-fastbook-an-html5-love-story

Charland A. & Leroux B. (2011) “Mobile application development: web vs. native”. *Communications of the ACM, Volume 54 Issue 5*, p. 49-53.

Claburn T. (2011), *Appcelerator Opens Mobile Developer Market*. InformationWeek 16 September, viewed 18 August 2014, from http://www.informationweek.com/mobile/appcelerator-opens-mobile-developer-market-/d/d-id/1100206

Dann J. (2012) *Under the hood: Rebuilding Facebook for iOS*, viewed 15 July 2014, from https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920

Du Q. (2012) *Under the Hood: Rebuilding Facebook for Android,* viewed 15 July 2014, from https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-android/10151189598933920

Fowler, M. (2011) *CrossPlatformMobile*, viewed 9 April 2014, from http://martinfowler.com/bliki/CrossPlatformMobile.html

Friberg, J. (2014) *Evaluation of cross-platform development for mobile devices*, Master’s thesis from Linköpings universitet, Institutionen för datavetenskap, Interaktiva och kognitiva system

Google Chrome (n.d.) *WebView for Android*, viewed 18 Aug 2014, from https://developer.chrome.com/multidevice/webview/overview

Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2013) “Evaluating cross-platform development approaches for mobile applications”. In *Web Information Systems and Technologies* (pp. 120-138)

Heitkötter, H., Majchrzak, T. A., & Kuchen, H. (2013) “Cross-platform model-driven development of mobile applications with md 2”. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 526-533)

Hemel, Z., & Visser, E. (2011). “Declaratively programming the mobile web with Mobl”. *ACM SIGPLAN Notices*, 46(10), 695-712.

Holder, E., Shah, E., Davoodi, M., & Tilevich, E. (2013) “Cloud Twin: Native execution of android applications on the Windows Phone”. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on ASE* (pp. 598-603)

IDC (2014) *Smartphone OS Market Share, Q2 2014,* viewed 18 August 2014, from http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Irish, P. (2013) *Fast performance CSS on mobile*, viewed 9 April 2014, from http://www.adobe.com/inspire/2013/05/irish-fast-css-on-mobile.html

Lehtimaki. M. (2013) *AppGyver Steroids Overview and Live Demo @ San Francisco 17th July 2013*, viewed 10 May 2014, from https://www.youtube.com/watch?v=oXWwDMdoTCk

MoSync (n.d.) *Native UI*, viewed 9 April 2014, from http://www.mosync.com/files/imports/doxygen/latest/html5/nativeui.md.html#Native%20UI

Mullany M. (2013) *5 Myths of Mobile Web Performance: Selected Slides*, viewed 18 August 2014, from http://www.slideshare.net/mullany1/5-myths-of-mobile-web-performance-selected-slides

Net Applications (2014), *Operating System Market Share*, viewed 18 August 2014, from http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1&qptimeframe=M

Olanoff D. (2012) *Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5*, TechCrunch 11 September, viewed 18 August 2014, from http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/

Polymer (2014) *Polymer and Web Components change everything you know about Web development*, viewed Aug 18 2014, from http://www.polymer-project.org/resources/video.html

Saggese, L. (2013) *Xamarin 2.0 vs Appcelerator Titanium vs PhoneGap*, viewed 20 May 2014, from http://stackoverflow.com/questions/17249500/xamarin-2-0-vs-appcelerator-titanium-vs-phonegap

Sommer A., Krusche S. (2013) *Evaluation of Cross-Platform Frameworks for Mobile Applications* in Software Engineering (Workshops) Pg. 363-376 2013

Tanaka, M., (2013) *Bringing Native UI to PhoneGap Apps | PhoneGap Day EU 2013*, viewed 5 June 2014, from https://www.youtube.com/watch?v=CidHx_Wh76E

Traeg, P. (2013) *Best Of Both Worlds: Mixing HTML5 And Native Code*. Smashing Magazine 17 October 2013, viewed 22 April 2014, from http://www.smashingmagazine.com/2013/10/17/best-of-both-worlds-mixing-html5-native-code/

VisionMobile (2014 Q1), *Developer Economics Q1 2014: State of the Developer Nation Q1 2014*, viewed 9 April 2014, from http://www.visionmobile.com/product/developer-economics-q1-2014-state-developer-nation

VisionMobile (2014 Q3), *Developer Economics Q3 2014: State of the Developer Nation*, viewed 18 August 2014, from http://www.visionmobile.com/product/developer-economics-q3-2014/

Wikipedia (2014a) *Multiple phone web-based application framework*, viewed 9 April 2014, from http://en.wikipedia.org/wiki/Multiple_phone_web-based_application_framework

Wikipedia (2014b) *Component-based software engineering*, viewed 9 April 2014, from http://en.wikipedia.org/wiki/Component-based_software_engineering

Wikipedia (2014c) *Cross-platform*, viewed 20 April 2014, from http://en.wikipedia.org/wiki/Cross-platform#Challenges_to_cross-platform_development

Wikipedia (2014d) *PhoneGap*, viewed 25 April 2014, from http://en.wikipedia.org/wiki/PhoneGap

Wikipedia (2014e) *Appcelerator*, viewed 25 April 2014, from http://en.wikipedia.org/wiki/Appcelerator

Wulf V., Pipek V. & Wun M. (2008) "Component-based tailorability: Enabling highly flexible software applications". *International Journal of Human-Computer Studies, Volume 66, Issue 1, January 2008*, p. 1–22

Xamarin (n.d.) *Components*, viewed 11 Aug 2014, from https://components.xamarin.com/

Appendix

Interview questions

A. Personal questions

1. Software development experience (e.g. “5 years of Java development”)
2. Mobile app development experience. Android?
3. View on cross-platform mobile app development today. PhoneGap? Xamarin? Problems, opportunities, future?

B. Questions for the Building Blocks App Builder web gui

1. Understood how to create an app? Event customization? Easy? Difficult?
2. Sufficient customization (events, methods)? To create any app you want.
3. Generate project files vs. fully compiled app? Important?
4. Any missing documentation (for app developers)? Improvements such as descriptions for methods and type checks maybe? Sufficient?

C. Questions for the Building Blocks Android framework

1. Understood how to create a component? Easy? Difficult?
2. Interfaces for component logic customization inside each platform (Eclipse, Xcode, etc) – is it enough? E.g. events & public methods?
3. Customization in IDE or web gui?
4. Documentation for Component developers - where? How? Source code, Java docs, website?
5. Your prediction: Market for components? Selling, free? Business opportunity? Who would create? Who would buy?
6. Integration between components – easy/difficult to create?

D. Final questions

1. App layout and similar in each IDE (e.g. Eclipse) – good/bad? Or force Component producers to add methods to the web gui for that?
2. Limitations? Limited to certain types of apps? Why/why not?
3. Opportunities? Faster development times? Better software?
4. Does the Building Blocks framework solve the problem with developing cross-platform mobile apps? Why/why not?
5. Final comment