MSc Software Engineering - Universiteit van Amsterdam

MSc Software Engineering

UvA Universiteit van Amsterdam

UvApp: Challenges in mobile software engineering examined

Xander N. Horjus

August 18, 2014

Supervisors: Vadim Zaytsev; Tom Kuipers; Alan Berg

Host company: University of Amsterdam ICTS

Abstract

We live in an information age, where in recent years the use of smartphones and tablets has exploded. In December 2013, there were more people with a smartphone than with a desktop computer. These smartphones are capable of quickly providing information over the internet, and due to their portability and ease of use favoured by many over using a PC. Therefore, what better way to provide students at an academic institute with study-related information than a smartphone application? Various institutes have preceded the University of Amsterdam (UvA) in creating and deploying such an application. However, mobile applications ("apps") are a relatively new phenomenon and field of research in the sector, and there are numerous challenges linked to their development process. In this thesis, these challenges are investigated and described. Furthermore, an information app for UvA students was constructed with these challenges in mind, using a framework called Kurogo, which describes itself as "mobile middleware". The aim of constructing this app was to provide the university with an information app for students, as well as researching whether the challenges mentioned in the common literature are still applicable when using such a middleware platform.

Contents

1 Introduction **3**
- 1.1 Background . . . 3
- 1.2 Research questions . . . 5
- 1.3 Thesis overview . . . 5

2 Development of Mobile Apps **6**
- 2.1 Native versus web . . . 6
- 2.2 Middleware . . . 9

3 Challenges in Mobile Software Engineering **10**
- 3.1 Fragmentation . . . 10
  - 3.1.1 Expected results . . . 13
- 3.2 Testing . . . 13
  - 3.2.1 Expected results . . . 15
- 3.3 Data usage . . . 15
  - 3.3.1 Expected results . . . 15
- 3.4 Security . . . 15
  - 3.4.1 Expected results . . . 16
- 3.5 Conclusion . . . 16

4 UvApp **18**
- 4.1 Introduction . . . 18
- 4.2 UvApp features and modules . . . 19
- 4.3 Implementation . . . 21
  - 4.3.1 Kurogo Middleware . . . 21
  - 4.3.2 Modifications . . . 22
  - 4.3.3 Modules . . . 24
- 4.4 Development timeline . . . 28

5 Results and Contributions **31**
- 5.1 Results: Frequency and extent of challenges in literature . . . 31
- 5.2 Results: New challenges found . . . 35
- 5.3 Contributions . . . 36
  - 5.3.1 Contributions to the field and the Kurogo platform . . . 36
  - 5.3.2 Contributions to the host organization . . . 38

6 Conclusion **40**
- 6.1 Acknowledgements . . . 41

CHAPTER 1

Introduction

1.1 Background

Looking at figures of smartphone ownership growth over the past decade (fig. 1.1), it immediately becomes apparent that we have entered a new era. An era of portable, connected devices. An era of "always-online". Smartphone ownership has grown so fast, that the smartphone market has overtaken the PC market by 2010, a mere 3 years after the first iPhone launched [34]. That year, smartphone ownership grew 87%, whereas PC ownership only grew 3% [34] [38].

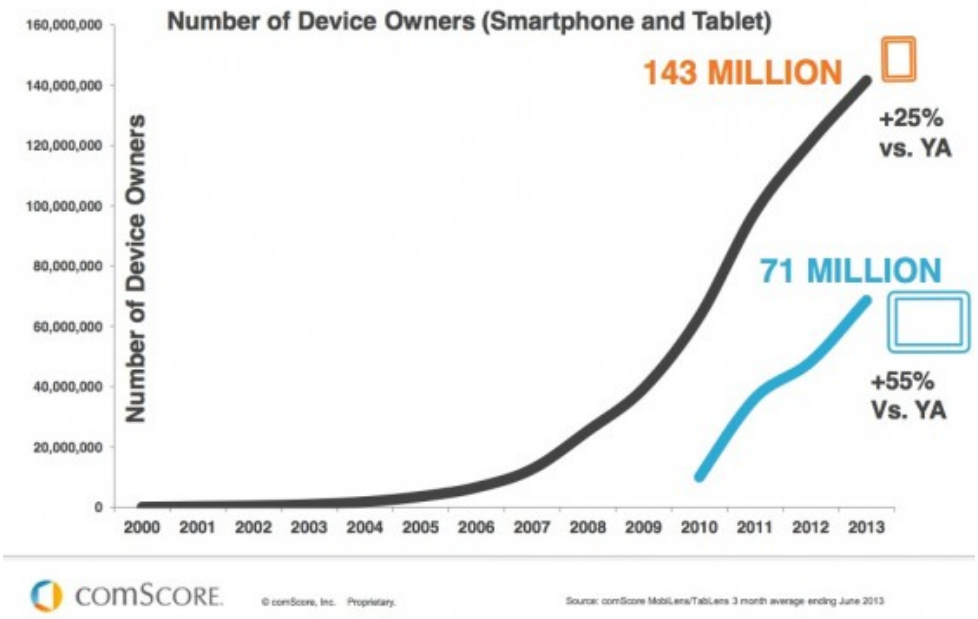

Figure 1.1: Smartphone and tablet ownership growth up to 2013

Surveys show that young adults in particular, ages 18-29, are avid smartphone users: 83% of all people in this age group use a smartphone [47]. These numbers indicate that smartphones are a great platform to provide information to (particularly younger) people on the go by means of an "App" (mobile application), no matter where they are. There is no need to turn on a computer and pull information: it can be pushed to the user's portable device, being able to reach them at all times.

Various universities, for example, have already deployed an App for students, which allows

them to quickly gain insight into whatever college-related data they require. A quick Google search for "university mobile site" gives over a million hits, of which several hundreds are unique mobile web applications for various universities (such as Harvard, Stanford, various U.S. State universities and so forth). An example of one of these apps found in literature is a case study by the Hong Kong Baptist University [52]. Another example is an app that was made for the Central Connecticut State University's Buritt Library [37]. The University of Amsterdam (UvA), did not have such an application at the time of this thesis' initiation (April 2014).

However, mobile application development is a relatively new field within the IT sector, and has only recently started to mature. There is still a lack of an extensive toolset similar to that available for traditional applications, which have existed and matured for decades. There are also various other challenges when writing mobile Apps, such as the large fragmentation of platforms and differences between mobile devices (i.e. smartphones and tablets). There are various popular platforms, such as Google's Android (61.9% market share), Apple's iOS (32.5% market share) and Microsoft's Windows Phone (3.5%) [8] [23] (market share numbers measured July 2014, and are the cumulative market share of smartphones and tablets). These platforms have varying philosophies, some open source and some closed source, and each of them has its own Software Development Kit (SDK), standards, advantages and limitations. However, fragmentation does not stop there. Within the same platform, there is more fragmentation. Whereas Apple's iPhone is produced only by Apple itself and is consistent with hardware architecture (thus ensuring that an iPhone App will always work for your iPhone, provided you meet the software requirements), Google's Android is an open platform with a large amount of manufacturers creating Android devices [27] [1]. Both hardware-wise and software-wise these devices have large variations (screen size, processor, RAM capacity, GPU, camera and so forth for hardware, and a large range of operating system versions for software [3]). Each of these variations need to be accounted for, and while the Android SDK does take care of some of these variations, App developers still need to take these into consideration during the development and testing process.

This platform fragmentation is not the only challenge. The technology that power smartphones pose several inherent problems. Issues such as finite battery life and ever-changing mobile network coverage and signal strength are to be taken into account. In most countries, mobile data is expensive and slow compared to household PCs. This means network usage needs to be minimalised. However, with limited flash hard drive storage, we can not simply store gigabytes of data on the device for a single App. In addition, with most smartphones having touch screens these days, how can we (properly) automate UI testing? The list goes on. Several pieces of software have been written to try and mitigate some of these these issues, but studying the literature and various blogs and forums on the internet shows that mobile development has far from matured fully, and many developers still struggle with these problems.

The purpose of this thesis is to gain insight into the state of challenges of mobile software development in 2014, by studying the challenges found in both scientific literature as well as informally mentioned on the internet. This will be done by constructing an information application for the UvA with the help of "mobile middleware" [1], which aims to minimise the effort and challenges that are inherent to the mobile field. The details and specific context of the application will be discussed in the following chapters.

[1] "Middleware"is defined by Wikipedia contributors as "computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue". Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application." - `http://en.wikipedia.org/wiki/Middleware`

1.2 Research questions

The main goal of the investigation is **to find out to what extent the major challenges in mobile application development can be overcome by the use of mobile middleware**. In order to answer this question and provide extra information, the following secondary questions arise:

- What are the typical challenges found in mobile software engineering?
- What are the trade-offs when using mobile middleware?
- What new challenges does using mobile middleware impose?

Due to a lack of literature concerning issues in mobile development, it is difficult to create a quantitative empirical experiment in which frequencies of challenges can be compared. There is a lack of real hard data to be found, which is logical considering the youth and fast developing nature of the field. An alternative experiment could be to create the same app twice - with, and without mobile middleware. These numbers can then be compared. However, even not considering other factors that could make the comparison unfair, the experiment is too large-scale for this master thesis due to limited timespan and manpower. This would lead to inconclusive results or an unfinished experiment. Therefore, we shall simply look at creating an application using mobile middleware, and compare the issues and challenges against those mentioned in the literature. We can't compare frequencies due to a lack of data, but we can compare the extent of issues in this single context (mobile information application for students, utilising middleware). Surveys held among developers (indicating how many developers find a particular problem actually an issue) and enumerations of problems in papers such as that of Joorabchi et al. [39] will be used as basis. The issues mentioned will be investigated, and how much of an issue they pose when using middleware noted and compared. Finally, any challenges not mentioned in literature will be noted as well.

1.3 Thesis overview

This thesis consists of various sections. It is structured as follows:

The reader is provided with background information about the mobile field and development of mobile applications.

Current challenges in the field of mobile software engineering (MSE) are then summed up and explained. The root cause of these challenges and why they are relevant are explained. Moreover, for each challenge, the expected results (i.e. can they be overcome by using mobile middleware) are given.

The application created for the UvA ("UvApp") is then detailed from all aspects: the background story, how the requirements came to be, the working environment and context, the development process and interesting findings.

Furthermore, the results of the investigation and the contributions by this thesis project to the field and to the UvA are outlined.

Finally, the conclusions which can be drawn from this thesis are described.

CHAPTER 2

Development of Mobile Apps

2.1 Native versus web

As previously mentioned, mobile applications have enjoyed an immense increase in use in the last 5 to 7 years. However, smartphones and "mobile applications" are a collective name for various devices and platforms, and there is no single standard. First, distinction can be made from two types of Apps: native and web. Native applications are installed to the device's operating system, similar to traditional PC applications. They run natively on the device, directly talking to the operating system by means of API calls. Web applications are deployed on a remote server, and are most often accessed by the mobile device's browser. Calculations are often made by the server hosting the application, with the app using web technologies to communicate with the server.

Both of these application types have their own advantages and disadvantages. Native applications are installed locally, and therefore do not require a connection to the internet to run, in theory (unless their functionality specifically requires a live internet connection, which is the case for most modern mobile apps). In contrast, web applications always require a working internet connection in order to even access the app. Native applications are limited only by the API and SDK that the platform provides and can use the device's full feature-set (such as extensive graphical display or use of notifications). Web applications are limited by the web standard and its implementation in the browser. While the latest web standard provides an increasing feature-set, it still is not up to par with native apps. Web apps have other advantages over native apps though: an update of the app is done server-side, which means the user does not need to do anything (i.e. specifically update their app through an App Store) to receive the latest version. Versions of the app are always consistent, as there is only one (the one hosted on the server). As the web standard and its implementation is relatively consistent, one implementation of the app can be served to all platforms (no need to create different versions for different platforms). Web development is also faster and often cheaper: there is no need to create multiple versions of the application, one for each platform. Instead, one web application can be written in any language of choice. There is no need for the developers to master multiple frameworks and languages, but rather can use their language of choice.

Furthermore, native applications are generally distributed and updated through a central service such as an app store [1]. They also need to be updated through this app store, and are often subject to the guidelines of the company that owns this app store (for example, the iPhone's

[1] An app store (application store) is an online portal through which software programs are made available for procurement and download

app store has a very specific set of guidelines which disallows violent or erotic material and so forth). While the option exists to manually download and install Android app packages (APK files), this is discouraged by Google and requires the user to specifically allow this in the settings a priori. With web applications, the entire application is hosted remotely and the app store is circumvented entirely. The user only needs to point their browser to the app's URL, and it will be displayed on screen.

Examples of native applications are games such as Angry Bird, but also popular apps such as Shazam (to identify music) or Skype (digital communication). Native apps can use the full range of services that the OS provides, detailed in figure 2.1.

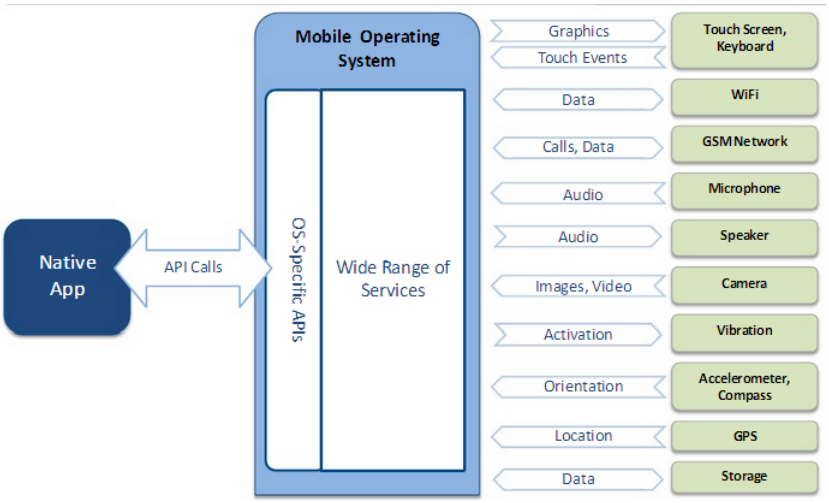

Figure 2.1: Interaction of a native app with the mobile device [28]

Examples of Web applications are the mobile YouTube page, the mobile Gmail client, the mobile Dropbox site and so forth. These are often web applications with a modified front-end tailored for mobile devices, but can also be separate applications written entirely for mobile (such as the Mobile Harvard site). Due to the front-end being written in HTML, which can be seen as a blank canvas, there is no enforced consistent look and feel. However, there are various toolkits out there (such as jQuery Mobile and jQuery Mobile UI [12], described as "A Touch-Optimized Web Framework") that use the latest HTML5 standard to bring interactive and responsive design to mobile web. Figure 2.2 describes the interaction between a web app and the mobile device.

On the rightmost side, the color of the box indicates the accessibility of the device's service. Green indicates a service is fully accessible, and the darkness of the red indicates how inaccessible it is. Microphone, camera and vibration are nearly completely off-limits to most web apps, whereas the other features (WiFi, Network, Speaker, Accelerometer, GPS and Storage) are either partly functional, or only on more recent devices.

However, there is a third type of application: hybrid. Hybrid apps try to combine both worlds, web and native. A hybrid application is a web app wrapped in a native shell. It is an installable binary, obtained through the platform's respective app store, which displays a web page when opened. In theory, it's best of both worlds: the binary can call upon the device's native services which web apps do not have access to, pass this information to the web server, and use the web app's other advantages. As the actual app inside is a web app, this saves production costs when developing for multiple platforms. Depicted in figure 2.3 is a visual representation of the interaction between a hybrid app and the mobile device [28].

However, hybrid apps are not the ultimate solution. They have the same caveats as web apps,

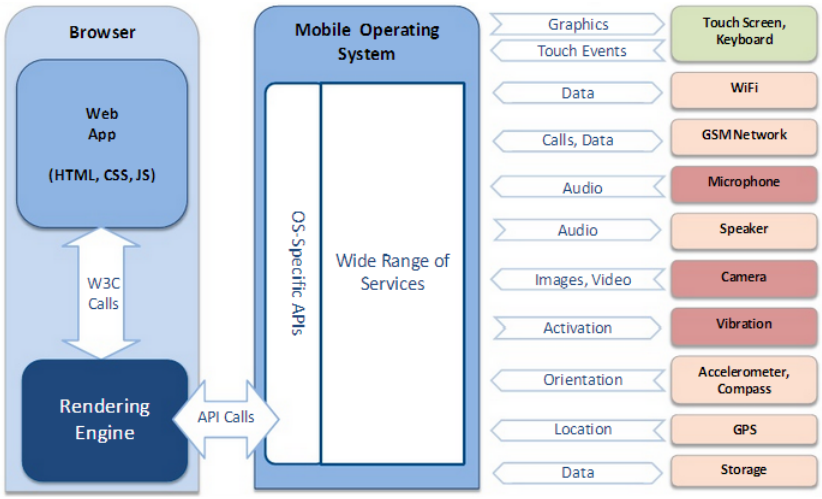

Figure 2.2: Interaction of a web app with the mobile device [28]

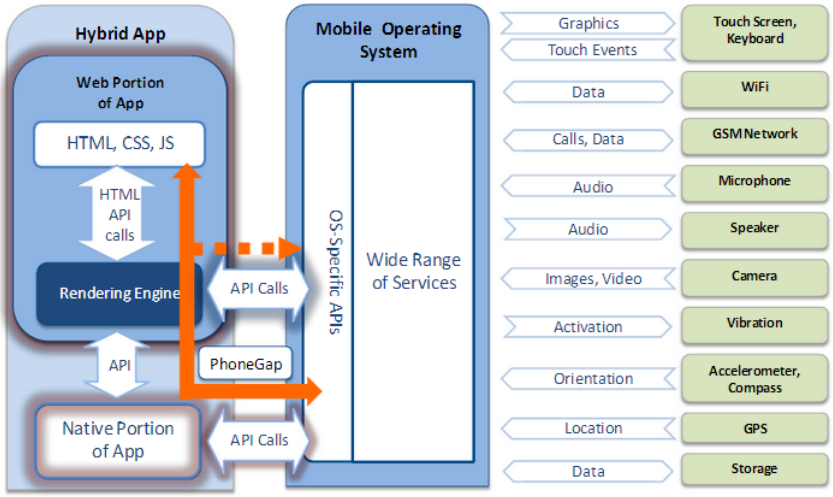

Figure 2.3: Interaction of a hybrid app with the mobile device [28]

in addition to also having the app store overhead. This can be an advantage or a disadvantage: if the part that uses native functionality needs to be updated, updates via the app store are still mandatory. In addition, as opposed to having no approval process, they need to be approved by the app store owners.

Concludingly, there are various type of applications: native, web and hybrid. Native acts more like a traditional software application, but web is easier, faster and more cost-effective for developers. Each have their own limitations and advantages. It seems there is no one size fits all solution for choosing the type of application; There are inherent trade-offs to be taken into account. Web apps have lowest time-to-market, lowest development costs and circumvent app stores, but they also have the worst user experience due to least feeling like an app and more like a mobile web site, developers say [39]. Furthermore, they have high response times due to needing to constantly communicate with a server. Native apps have a far better user experience and more capabilities, but also higher time-to-market and high development costs. Smutny (2012) remarks "*if the mobile application is mainly used to display and interact with online content or services, it is better to avoid the native choice and go for web*" [49].

2.2 Middleware

In order to assist developers, several solutions have been created to optimize development for mobile. "Mobile middleware" are software development frameworks which form a layer between the device and the application, and generally apply to web apps. They allow developers to focus on developing functionality, rather than the intrinsic characteristics and limitations of the various platforms. Techopedia defines mobile middleware as "*middleware used in the context of mobile-computing devices Mobile middleware offers various transparencies that hide the complexities of mobile environments. For instance, location transparency allows applications to exchange data with other applications without any regard for their location. Similar abstractions are likewise provided by transparencies on the transport protocol, operating system, programming and others. Mobile middleware typically involves services like messaging and Remote Procedure Call (RPC), resource discovery, transactions, directory, security, storage services and data synchronization.*" [24]. In sum, middleware allows developers to write code once and deliver a consistent experience on multiple platforms. However, the various limitations of web applications still apply.

Examples of current middleware platforms are Kurogo, PhoneGap, Accelerator, RhoMobile and appMobi. Comparing each of these middleware platforms is outside the scope of this thesis and has been done before [49]. However, it is noteworthy what they have in common: support for the major mobile operating systems, bringing a consistent look-and-feel, using the HTML5 and CSS3 standards for interactive design, a small resource footprint, a large library for common app functionality, a touch/mobile-optimized interface and the possibility to wrap the web app in a native binary (i.e. generate a hybrid app).

CHAPTER 3

Challenges in Mobile Software Engineering

3.1 Fragmentation

There is a lot of fragmentation in both native and web applications [39], both across platforms and within the same platform. User interfaces, user experiences, user expectations, difference between device specifications (screen size, computability power, hardware features) and a large variance in operating system versions are all examples of this fragmentation. Joorabchi et al. [39] report that mobile devices are actually moving more towards more fragmentation rather than unification; new phones and operating system versions are coming out faster than old phones are discarded.

Purely looking at phone devices, the following figure shows a representation of the most popular Android devices and their market share by size [20] as of July 2013:

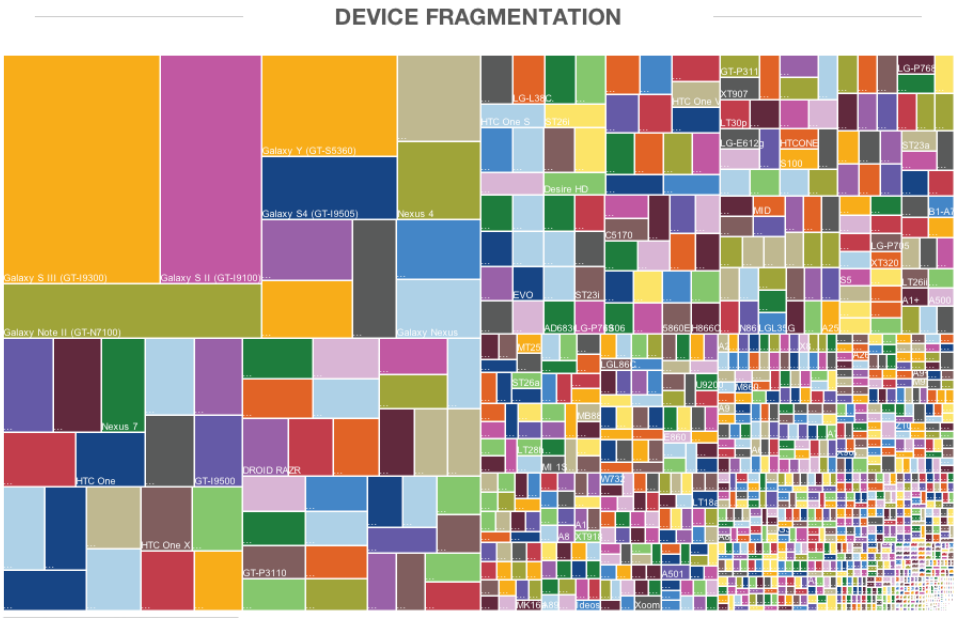

Figure 3.1: Android device fragmentation

Despite the information itself being somewhat outdated, with new phones having been released since, the figure clearly depicts the sheer amount of different Android devices – each with their unique hardware profile. Table 3.1 shows information about platform versions for Google's

Version	Distribution
< 2.2	approx. 1%
2.2	0.7%
2.3	13.5%
4.0	11.4%
4.1	27.8%
4.2	19.7%
4.3	9.0%
4.4	17.9%

Table 3.1: Android version distribution

Android [3].

Despite being simply "minor" version releases (0.1 increments), new functionalities are incorporated in nearly every release and old functions are deprecated from the API. Nearly every version has to be taken into account when developing for Android. For instance, due to the new notification system introduced in 4.1, an app that works on that version might not work on 4.0 at all. There is, effectively, not one but 7 operating system versions to account for.

Further fragmentation can be found in screen sizes and densities [1]. The following graph depicts the fragmentation in screen sizes and densities:

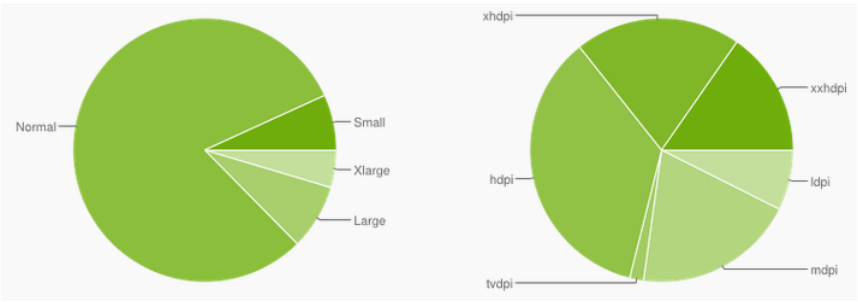

Figure 3.2: Distribution of Android screen sizes and densities

There are four different screen sizes, and six different screen densities to take into account. For each of these sizes, a different GUI layout is to be created for every screen view in the app. A developer who builds a view for an extra large screen and completely fills it will have to make compromises in order to make the app accessible for smaller screens.

This is a considerable challenge that has to regularly be dealt with, report the developers surveyed by Joorabi et al. [39]. The Android Developers website states: "Generally, it's a good practice to support about 90% of the active devices, while targeting your app to the latest version." [4]. The website provides some guidelines that aims at minimising the effort to provide the largest compatibility, but it nevertheless is an issue that needs to be constantly regarded.

Apple's iOS and Microsoft's Windows Phone deal with these issues to a lesser extent. Whereas Android has tens of manufacturers creating phones [27] [1], each with their own hardware specifications and varying architectures, Apple has a monopoly on iOS devices. iOS devices have a far more consistent hardware setup, with Apple ensuring that nearly every device can install their latest updates. Furthermore, they push these updates fast and in such a manner that users will be inclined to update. This shows in their usage statistics [6]: 90% of devices are using iOS 7, their current most recent version, 9% is using iOS 6 and a mere 2% is using earlier versions.

[1] Screen density is defined as "the quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch). For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen" [2].

This means targeting the latest version already gives a 90% coverage, compared to Android's latest version being installed on only 17.9% of devices. Moreover, the iPhone has only had two variations in screen size since its launch: 3.5" and 4". However, iOS imposes other limits: being a closed platform, Apple has a firm grip on what it allows the developers to access on the OS level, and this is considerably more strict than what Android allows [39]. This causes developers to sometimes be forced to look for backdoors to obtain certain functionality, which poses a further challenge. However, the challenge goes the other way: as Android is open source, each manufacturer modifies the source code to their own desire. Sometimes these changes are not compliant to the standards, which creates more complexity [39].

Why is fragmentation actually a problem? It seems logical that mobile developers would not want to exclude a large part of their target audience. Upon choosing to develop a native app, for instance due to needing native features, developers in practice will have to develop for both iOS *and* Android. This, in turn, implies they need to take into account two major platforms (each with their own programming language and vastly different SDK), 10 operating system versions, various screen sizes and so forth. Reusing code is difficult; Joorabchi's survey states: "*The majority of developers stated that it is impossible or challenging to port functionality across platforms and that when code is reused in another platform, the quality of the results is not satisfactory*" [39]. In order to assure user satisfaction, an app must work (mostly) properly on all the various combinations of device hard- and software. However, if the developers know that their target audience is for instance exclusively iPhone 5 users with iOS 7, they can focus fully on this one composition, and fragmentation is not an issue. Unfortunately, this is hardly the case in practice, where commercial app developers will want to make their app available to the largest audience possible.

Web apps solve some of this fragmentation. Especially the latest HTML standard, HTML5, is designed with mobile capabilities in mind [32]. It allows the developer to use device features that were previously limited to native apps, such as geo-location and accelerometer data. HTML5, as opposed to prior versions of the markup language, has considerable support for interactivity and in combination with Javascript can provide full client-side services that approach native applications [48]. However, the standard is still under development, and there is also fragmentation between browsers and their support of the various HTML5 features (see figure 3.3).

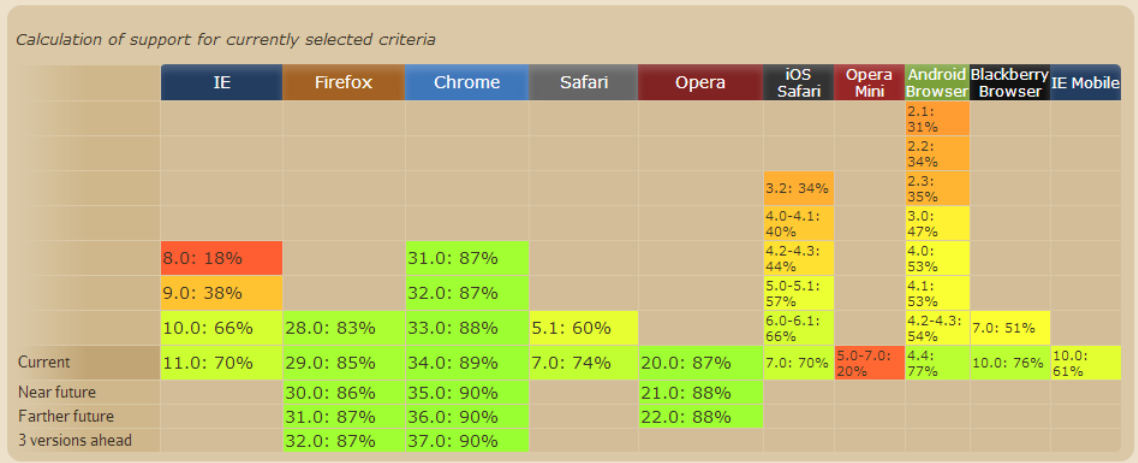

Figure 3.3: Calculation of HTML5 feature support per browser version, June 2014 [9]

These fragmentation issues are a challenge not only for the development process, but also for testing [39]. Apps need to be tested against a large range of system versions and screen sizes. This makes testing a very time-consuming and expensive process.

To conclude, fragmentation is a significant challenge for developers. The large range of hardware compositions, software versions and various platforms create a high amount of complexity for development and testing. Even when creating a web app, moving from the limitations of native apps to the unity and limitations of the web standard, there are multiple browsers with different versions to consider.

3.1.1 Expected results

Is the "fragmentation" circumventable or fixable by using mobile middleware? Most likely not fully. Part of the issue of fragmentation is having to develop for multiple platforms and versions, each with their own API and methodology. This can already be circumvented by simply building a web app rather than a native app. Considering the current mobile middleware frameworks are used to develop web apps (with the possibility to generate native wrappers), at least part of it is circumvented. Frameworks such as PhoneGap [21] generate a native wrapper for each of the platforms programmatically, and thus not even the native wrapper has to be made manually.

However, having moved into web app territory, web app characteristics and limitations apply. There is still fragmentation among the major browsers when it comes to HTML5 support, something to take into consideration. Moreover, there are still the various screen sizes to take into account. This is something the browser largely does for the developer, however still something to consider. If being a web app rather than native is not an issue, for instance due to not needing the features that web apps do not support and not needing the native speed, it is expected that mobile middleware does indeed solve a large part of this problem.

3.2 Testing

Another challenge mentioned in literature is the lack of automated testing for mobile devices [39]. Mobile apps, native specifically, have not existed for very long and have evolved fairly quickly. Whereas the devices and apps have quickly grown in power and complexity respectively, emulators for the major platforms have remained behind, not supporting important features for testing such as mobility, location services, sensors or different gestures [39]. Mobile applications further add complexity to testing compared to traditional software, due to factors such as energy consumption. In practice, battery life is a valuable resource for mobile users. The extent of battery drainage from an app is determined by a lot of factors, some of which are nearly impossible to know a priori, due to for instance being caused by inter-application communication [51]. These issues will not show up on emulators and are difficult to test. Other dynamic factors that can influence bugs are the connectivity quality and device operational mode [45].

Hardly any native app testing tools were available until recent years due to the novelty of the platforms [36]. Another implication of this novelty is that developers are less familiar with the platforms and their constraints such as low-power CPU and small memory [36]. Studies confirm these allegations: Maji et al. found that Android applications can have bug densities orders of magnitude higher than the OS itself [43]. Most of the literature on automated Android and iOS testing is focussed on GUI bugs [31] [36] [30] [50] [54] [40] or permissions security [35] [42] [53], and this reflects on the available testing suites. Automated test suites are becoming increasingly more available, with suites like Appium [5] and Experitest [10] allowing automated testing on all platforms. However, these are largely focused on GUI testing. Other classes of bugs, such as unhandled exceptions, API errors, I/O errors or concurrency errors have considerable fewer testing options.

Moreover, the earlier mentioned fact that emulators lack support for features such as sensors, make some aspects of automated testing impossible. These imperfections reflect on practice: a

survey by Joorabchi et al. [39] shows that only 3.2% of developers fully automate their testing (see figure 3.4). Interestingly, looking at levels of testing and respective methods, automated GUI testing is done very little in practice (see figure 3.5), despite the relatively overwhelming amount of research and availability of tools compared to other test levels. Unit testing is still most commonly automated, which is somewhat obvious as automated unit testing is easily implemented and is well integrated into most developers' processes.

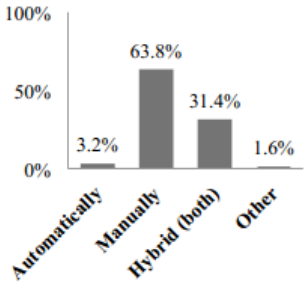

Figure 3.4: "How are your native mobile apps tested?" [39]

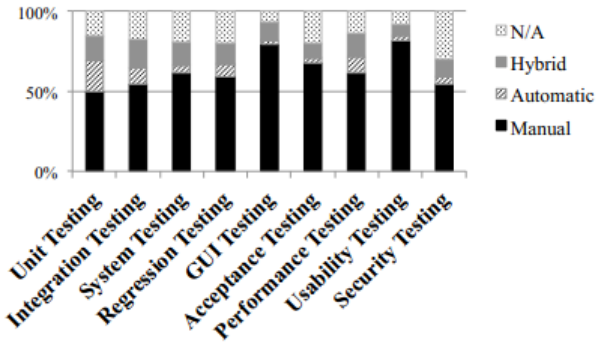

Figure 3.5: "What levels of testing do you apply and how?" [39]

Why is this a problem? As Muccini et al. [45] note, the largest factor is efficiency. Automated testing is cost-effective: it is faster than manual testing (time savings translate into cost savings) and it requires little manpower. Furthermore, automated testing has the possibility to simulate situations that manual testing can not (for instance, in case of a client-server structure, thousands of simultaneous requests).

However, most - if not all- of the aforementioned issues, are applicable to native or hybrid apps only. Mobile web apps behalve like regular web apps, and their testing situation is mostly identical to that of regular web apps. Some features, such as accelerometer data, are exclusive to mobile devices and an exception to this.

To conclude, automated testing is difficult on mobile devices due to a lack of complete testing functionality, lacking emulators and due to external factors that will influence the app but are difficult to reproduce in a test setting, such as inter-application communication. In addition, the test possibilities that exist mostly focus on GUI bugs or permissions security. While these are important aspects, they do not nearly cover all classes of faults. However, nearly all of these issues apply exclusively to native apps. Testing of mobile web apps closely resembles testing traditional web apps, with the added complexity of unstable connections (see also section 3.3).

3.2.1 Expected results

As priorly mentioned, the issues regarding the automation of testing nearly exclusively apply to native apps. As mobile middleware creates mobile optimized web apps, these issues are inherently circumvented. For web apps, regular web app testing suites can be used. The remaining issue is lack of automated testing for newer, mobile-specific HTML5 features such as accelerometer.

It is expected that the mobile middleware will not overcome this challenge in the near future due to a lack of automated testing tools for these particular features. However, when HTML5 matures and automated testing tools for these features become more available, mobile middleware packages will most likely incorporate this, overcoming this challenge nearly completely. However, it is expected there is one added challenge related to testing mobile web apps that is not present in traditional web apps: mobile data connectivity tends to fluctuate. It varies from fast, stable connections, to slow, unstable connections. These issues associated with transmission through gateways and the telephone network add a layer of complexity to mobile web apps [51].

3.3 Data usage

Another challenge found in literature is related to connectivity. Briefly mentioned in the previous section, mobile devices tend to have unreliable data connections, which is an issue for developers [39]. Connections might (seemingly) randomly drop as signal strength declines or the mobile device switches access points or phone masts. However, even not considering the variable stability of mobile data networks, mobile data is often expensive. Mobile service providers might charge their customers based on the amount of bandwith used in a given timeframe.

Needless to say, dealing with data is tricky for data intensive apps [39]. Transmitting a large amount of data over the network is difficult due to its unreliability and possible costliness for the user. In addition, mobile devices have limited storage capability. As of mid 2014, high end smartphones tend to average 16 to 32 GB of non-expendable flash drive storage capacity [7] [19] [11]. Apple's iPhone 5S is available in three models: 16 GB for €699, 32 GB for €799 or 64GB for €899. A 16GB upgrade for €100 (€6.25/GB) or a 32GB upgrade for €100 (€3.13/GB). In 2014, PC hard drives cost an average of €0.05/GB [29]. These numbers indicate how costly storage capacity is compared to traditional desktop environments. While there are certain devices that have an SD-slot which allows the user to extend their storage capacity [22], the majority of current devices simply do not [7] [19] [11]. It is clear that storage capacity is valuable on mobile, and apps cannot simply store gigabytes worth of data on the device. However, as priorly mentioned, it is also not an option to transmit large amounts of data over the network. Developers must keep these constraints in mind and try to construct data- and network efficient solutions.

To conclude, mobile data transmission is costly and unreliable. In addition, storage capacity is costly on mobile devices and should be used sparingly. Developers should try to write lightweight applications that use data and network efficiently and effectively.

3.3.1 Expected results

It is expected that mobile middleware will not overcome this challenge. This challenge is inherent to the mobile platform and can not be circumvented by software.

3.4 Security

Another major challenge in mobile software development, which has nearly become a field of its own, is mobile security. Mobile devices have become sophisticated and are laden with personal

data, such as contacts, e-mails, stored passwords, instant messaging communications and even bank account information. As a result, mobile devices have become tempting targets for attackers and malicious software [46], for the following reasons.

First, it allows attackers to generate revenue [33]. For instance, by invoking events that are billed by mobile network operators or using payment systems for which the mobile device is used as an authenticator. Furthermore, the invasion of privacy and accessibility of personal information. E-mail addresses, bank account details, even contacts, are interesting data for an attacker.

Mobile devices are often attacked by one of the following methods [41]. First, by having the victim click on a link which uses an exploit in the web browser to install malware. These are often placed by infected users through their social networks, which are deemed trustworthy by others. Another method is the use of unencrypted Wi-Fi hotspots, which attackers use for man-in-the-middle attacks. Subsequently, they can intercept and control the user's traffic [41]. Phishing links as often found in desktop environments are another possibility. Finally, specific apps can contain security exploits. While software updates can be enrolled that fix these exploits, users often do not install these updates, leaving the software vulnerable [51].

Traditional anti-malware approaches often involve antivirus and firewall products. However, these are obstructed by the aforementioned constraints: the lack of processing power and scarcity of battery life. Complex malware detection algorithms may work well in desktop environments, but on mobile they are less effective and more costly [46]. In addition, due to restricted OS kernel access on the mobile device, programming limitations are imposed that make the software expensive to develop.

To conclude, due to mobile phones increasing in popularity and containing rich data, they have become enticing targets for attackers. Traditional security is difficult due to the hardware constraints of the devices, and users are often security-unaware, not keeping their possibly vulnerable apps up-to-date.

3.4.1 Expected results

For this particular challenge, it is expected that mobile middleware can be part of the solution. Some vulnerabilities are on the system level and can not be fixed by any external software (i.e. exploits in the OS) whereas others are on the user level (clicking on a phishing link). Mobile middleware can not prevent the user to click on an infected link outside of the app. However, mobile middleware can be used to inform the user that they are leaving the site, or are possibly clicking on a dangerous link. While middleware can not do anything to prevent malware directly, it can assist in informing the user and making them aware of security risks on the device.

3.5 Conclusion

To conclude, it is clear there are numerous major challenges that mobile software developers face.

Most developers report that fragmentation issues are the most challenging [39] [51]. These fragmentation issues are caused by the great amount of unique devices (each with their own specific hardware compositions), as well as the amount of different software versions (each with their own API).

Furthermore, testing is more difficult on mobile than in desktop environments due to factors that influence app behavior which are hard to reproduce in a test setting. However, testing is also made difficult by the novelty of the field, which has not yet allowed testing tools and emulators

to mature. Not all device functionality can properly be tested, and not all classes of bugs are yet supported by these tools.

In addition, mobile apps tend to be reliant on internet connections while mobile networks themselves are not reliable. These networks tend to be slow and expensive. Device storage capacity is often also limited, which means both data transmission and data storage should be used as sparingly as possible. For developers, this means writing efficient apps.

Finally, security is a major challenge. Software is generally not allowed access at the kernel level, and coupled with the low-power CPUs in mobile devices, traditional antivirus algorithms do not scale well on mobile. Users are also often not security-aware, keeping vulnerable apps unpatched.

CHAPTER 4

UvApp

4.1 Introduction

In order to fully investigate the challenges in mobile software engineering and whether they can be overcome by the use of mobile middleware, an app was constructed for the University of Amsterdam (UvA). This chapter details its specifications and other relevant information regarding this app.

First, it is worth giving some background information about how the idea for this app came to be. Over 80% of young adults (ages 18-29) has a smartphone, an age group most students belong to. This thesis project was hosted by the UvA, who has no real go-to mobile information app for students. Such an app could be used to provide students with study information (grades, timetables and so forth) from their mobile device. There exist several information systems for UvA students, such as Blackboard and SIS. However, neither of these provide a mobile-friendly interface. Moreover, there are some fragmented interfaces for several other pieces of information: there is a "mobile grades" site [1] and a UvA student portal called MijnUvA. Nevertheless, there is no "one stop" mobile information portal. The UvA's ICT Services department (ICTS) wanted to develop a proof-of-concept for such an app, which connects all data sources into one mobile friendly overview.

Consequently, a joint project was set up where the UvA mobile app (UvApp) would be developed, whilst investigating the challenges in mobile software engineering for this master thesis. However, in order to get a good idea of what students truly find desirable features in such an app, the UvA's Faculty of Humanities (FGw, for *Faculteit der Geesteswetenschappen*) was also involved. In sum, the stakeholders in this project were the ICTS and FGw.

However, due to developing an information app, several constraints were inherently imposed upon this research. As mentioned in section 2.1, Smutny et al. (2012) rightfully note that if the app is mainly used for display and interaction of online content or services and the full speed of the device is not required, it is better to go with a web app rather than native [49]. Especially in the case of UvApp it seems the logical choice: the student information systems that it retrieves its data from are all hosted by the UvA and accessible only from the UvA network. Moreover, the target audience is expected to use a wide array of devices, and a web app is most cost- and time effective. It also circumvents overhead such as forcing students to invest effort to install the app from an app store and keep it updated. Considering this project was to done by one programmer in 3 months, a web app seems the best choice.

Unfortunately, this implies that several aspects and challenges that are inherent to native apps

[1] `http://m.sis.uva.nl`

can not be tested, as they are circumvented by using web technology. Furthermore, the mobile middleware used (Kurogo by Modo Labs [18]) does not allow generation of hybrid apps without a commercial license. Therefore, the native and hybrid parts of mobile software engineering are disregarded, and the focus was put on the challenges in mobile web app engineering. However, should the UvA decide to work with UvApp, it is recommended to purchase the commercial license.

In the following sections, the specifics of the app will be detailed: the features and modules, but also implementation details and technical documentation.

4.2 UvApp features and modules

Early May 2014, the author of this thesis sat with one person from the ICTS and two from the FGw to brainstorm about the UvApp. A list of desirable features, grouped by "must-have" and "could-have", was one of the subjects of this meeting. Considering the limited time span available for this project, it was imperative to agree upon a list of features that was considered a bare minimum (i.e. the must-haves). If, at the very least, these features were satisfactorily implemented, the project could be called a success. Moreover, a list of features that would make the app more attractive, but not necessarily required for a successful project, was created (i.e. the could-haves). Eventually, the following list of functionality was decided upon.

The following features were classified as "must-haves":

CAS login functionality

Students and staff should be able to log in via the UvA's Central Authentication System (CAS). This is the same authentication system nearly all UvA services use. Using this system has several advantages. First, it means that if a user was already logged in to another UvA service, they will not have to login again that session. Similarly, if they login via this app and subsequently open another UvA web service, they will already be logged in, reducing the amount of actions a user has to perform. Another advantage is that all UvA students and UvA staff will already have an account, rather than having to register and remember new credentials. Finally, as CAS uses UvAnetID (unique identification code) for usernames, a successful login instantly returns the person's UvAnetID, which can then be used to retrieve personalized data (for grades or Blackboard modules, for instance).

Blackboard announcements

The app should be able to retrieve Blackboard announcements for a student's courses. In one overview, students should be able to see recent announcements for every course they are enrolled in. This was considered one of the most essential features, as there is currently no mobile friendly interface for Blackboard, while Blackboard is the most widely used information system by UvA students, according to the FGw.

Grades

In addition, the app should show a student's grades. These grades can be obtained from SIS, which is where grades are entered when they are final. There is currently a simple, mobile-friendly interface for exactly this information at `https://m.sis.uva.nl/studieresultaten`. However, this is apparently not a frequently visited site (according to the FGw). For teachers, this module does not return any results.

Available study locations

The UvA provides its students with several locations where they can study and/or use PCs. These "studieplekken" or "studiecentra" (study centres) have a limited amount of seats and are open at specific times. There are several resources the UvA uses to provide its students with this information [2], but these are fragmented. Moreover, there is no mobile friendly overview for this information. A button to navigate to a certain studycentre via the device's GPS could be incorporated.

Timetable

The timetable page should show a student's upcoming courses and events, as well as provide a link to the mobile UvA timetable site [3].

Course enrollment

The app should provide a link to, the course enrollment interface located at `https://m.sis.uva.nl/vakaanmelden`. It is not desirable to create a new interface for this, as it is a relatively complex interface which should not be duplicated. Rather, the FGw and ICTS would keep the existing interface for this. However, a link to this site should be incorporated in the app with high visibility (i.e. in the menu bar).

Multi-language support

With the high amount of international students the UvA has, the app should be available in both English and Dutch. The default language should be set to Dutch. As "could-have" addition, it could be implemented in such a way that allows easily extending the amount of languages (for instance, by translation files).

The following features were classified as "could-haves":

University library catalogue

The university library (or UB as it is colloquially referred to, for "universiteits bibliotheek") has an online search tool. The entire catalog can be sought through, with all meta-information directly shown and direct links to fulltexts if available. The app could incorporate either a link to the UB catalogue search tool, or incorporate an entire interface (i.e. display the results in-app as well). Despite trying to avoid duplication of interfaces as much as possible, this was specifically requested by the UvA's mobile expertise team.

Social media

The UvA has various social media accounts, such as *UniversityofAmsterdam* on Facebook and *UvA_Student* on Twitter. Feeds for posts from these accounts, with possibly links to directly post to these pages/accounts, could be shown on the app.

Navigation and public transport routes

The various UvA locations are scattered throughout the city of Amsterdam. It could be interesting for students to get navigational directions from their mobile device to one of the UvA locations, by use of GPS and a service like Google Maps. However, another possibility is to use 9292OV [4] to see what the available public transport routes are from the user's location.

[2] Main site: `http://student.uva.nl/az/content/studiecentra/studiecentra.html`. Locations: `http://uba.uva.nl/locaties/?tab=2`. Available PCs: `http://uba.uva.nl/locaties/?tab=4`

[3] UvA timetable site located at `http://rooster.uva.nl`, mobile version at `http://rooster.uva.nl/m/`

[4] A Dutch service that helps the user move between locations using public transport, giving departure and arrival times and connection information.

Contact details UvA staff

Occasionally, students will find that they are in need of contact details for UvA staff, most often lecturers. A search tool to type in a lecturer's name and retrieve all relevant contact data (such as phone number, e-mail address and so forth) could be very useful.

News

The UvA has various news feeds, which could be shown on a News page.

Links to various resources

The app could have a module with links to various UvA resources. Examples of these are: "Studiegids" (course catalogue), SIS, Blackboard, MijnUvA and UvA Webmail.

4.3 Implementation

The following chapter describes the process and specifics of implementing the code for UvApp. First, the implementation of the Kurogo middleware will be outlined: how it is set up, how modules are written and how it can be extended. Next, modifications to the Kurogo platform in order to add functionality that was not supported initially is described. Afterwards, a timeline of the 3-month development period will be given. Finally, documentation is provided which assists in understanding and configuring the system.

4.3.1 Kurogo Middleware

The difference between the various types of apps was described in chapter 2 and an introduction to mobile middleware was given in section 2.2. In this section, the Kurogo middleware specifically shall be described.

Kurogo [5] is mobile middleware created by Modo Labs [6]. Modo Labs describes Kurogo on its website as:

> "Kurogo is open-source Mobile Optimized Middleware for developing content-rich mobile websites and iOS and Android apps. Created by Modo Labs, Kurogo emphasizes extensibility, clean integration, and exceptional UX. It powers the mobile presence of a broad range of institutions, from top universities to Fortune 500 companies."

Kurogo is an open-source PHP platform used by tens of universities around the world [15]. This shows when looking at out-of-the-box modules: "Courses", "Transit", "Admissions", "Library", "Social" and "Athletics" to name a few. The platform has most of the standard modules that a university would want in an app, readily available. However, its main power lies in that it is a platform which caters to mobile users. There are native iOS and Android libraries to generate hybrid apps (as mentioned in section 4.1, these will not be used for this project), the platform is lightweight - leaving a memory footprint of only several megabytes - and thus fast, and there is built-in detection for various types of mobile users. Differentiation is made between basic devices such as small screen feature phones, simple touch devices, compliant devices (i.e. Webkit-based smartphones) and tablet devices [16]. For each of these, the presentation is adapted to best fit the device. Basic devices will get simple, light-weight pages with minimal makeup, whereas tablet devices will receive more advanced pages (i.e. full Javascript/HTML5, high dpi images and more content due to higher screen resolutions).

[5] http://www.kurogo.org
[6] http://www.modolabs.com

Moreover, the framework has a solid code structure representing the Model-Viewer-Controller (MVC) structure. So called "connectors" act like models to retrieve data from the various data sources. This data is then parsed by a relevant parser, which converts the data into a PHP array or other useful format. Subsequently, the web- or API module handles the data with logic, which produces a view which is passed to the templating engine. The templating engine detects the users device (basic, simple, compliant or tablet) and places the view inside the relevant presentation template. Figure 4.1 describes this structure. Each Kurogo module has an `ini` file with module details. These contain the module's name, whether it is enabled, feed locations and custom options.

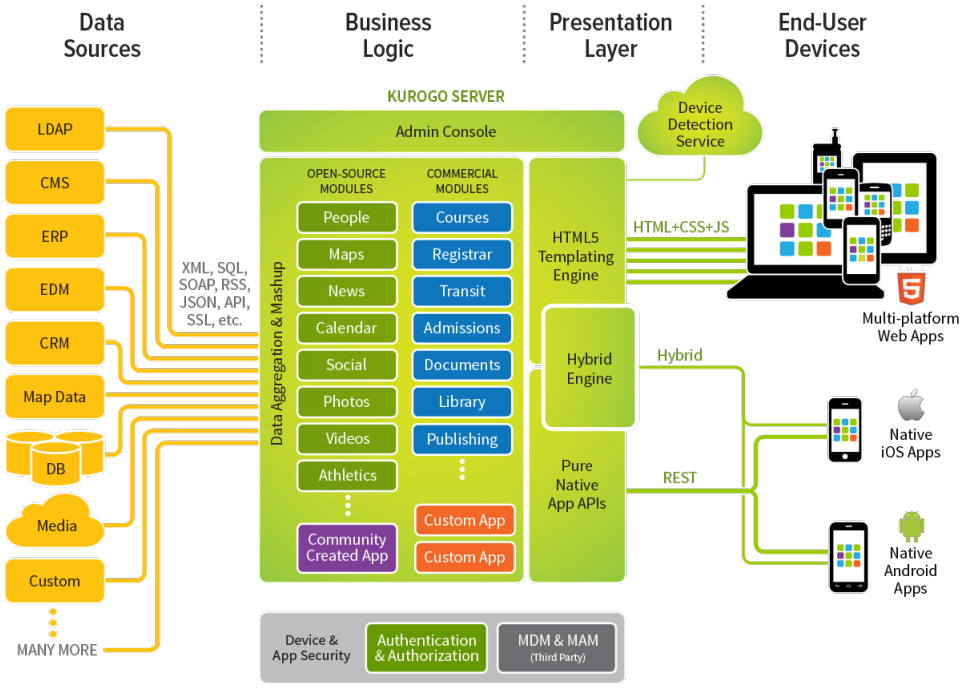

Figure 4.1: The Kurogo platform [18]

In addition, part of Kurogo's power lies in its vast library of connectors. There are out-of-the-box connectors for LDAP, ICS, RSS and so forth. Furthermore, data is nearly always obtained in one of several conventional formats: most often XML or JSON. The parser auto-detects the source format and converts it to a PHP array, ready for the logic to handle. This means simple modules can be created by simply pointing at a feed URL (of, for instance, an RSS or ICS feed) and letting Kurogo do the rest.

4.3.2 Modifications

Although the Kurogo middleware has an extensive out-of-the-box featureset, there were some requirements for the UvApp which were not supported by Kurogo without modifying the middleware itself. Although it was attempted to modify it as little as possible in order to keep maximum compatibility with future versions, some features inevitably required changes to the Kurogo core code. These changes are documented below. In addition, several fixes were made to the Kurogo system which were not applicable to UvApp only, but possibly to all Kurogo users. These modifications are described in section 5.3.

Administration console [7]

The Kurogo platform comes with a built-in administration panel, accessible via the `/admin` request. This administration console allows to set system configuration values, as well as module configuration values. These values vary from string values for module texts (titles, help texts, etc.) to feed paths or URLs, to the maximum amount of search results and so forth. As most of these configuration values are stored in the module's respective `ini` files (module.ini, feeds.ini and so forth), this administration console enables these settings to be changed while not granting access to the file system. As Kurogo offers no other possibility to change these values other than changing the `ini` files, this is a very convenient feature. However, the Kurogo documentation mentions the following:

> "This module is only meant as an aid to speed up initial configuration. It is not intended for production configuration and does not expose all values necessary to configure certain modules." (Kurogo Documentation, mid 2014. [13])

Modo Labs indicates that the purpose of this module is to speed up initial configuration, as opposed to keeping the module active during production and allowing to change configuration values. It can only be assumed that they chose to deploy it in this manner due to their assumptions that:

- Once the web application is deployed, it needs to be configured no further. There are no changes down the line, or any changes to be made should be done by the administration module in the development environment, with the altered `ini` files then transferred to production;
- Alternatively, any person that is authorized to change configuration values also has filesystem access.

Both options imply that any change to be made requires file system access, to either directly change the `ini` or to transfer an altered `ini` to the production environment. However, there are multiple scenarios where these restrictions are not applicable. For instance, an example configurable setting is the age limit and amount of results to be shown. We should be able to authorize any user to change these settings (for instance, we want the age limit of results from 14 to 30 days), even though we might not want to grant them full filesystem access, which allows them to manipulate the database or look at LDAP passwords stored in a different `ini` file. Unfortunately, Kurogo does not support this out of the box.

Therefore, a simple authorization system was built into the platform. As the LDAP based CAS authorization system does not return any information about user roles (i.e. staff or student), an additional `ini` setting was created in the admin panel. This setting is a comma-seperated list of CAS user IDs that are allowed to access the administration panel. These UIDs can be altered in the administration panel itself. In order to prevent hijackings or accidental removal of access for *all* administrators, a "root admin" value is stored in addition. This root admin ID is only editable by means of directly changing the `ini` via the filesystem.

With these changes, the administration panel can be kept enabled in production, and used to change less crucial values that do not need to be tested in a development environment.

Per-session internationalization [8]

One of the desired features was to make the app available in both Dutch and English, for both local and exchange/foreign students. As Kurogo advertises with 'internationalization' options, this seemed to be an easy feature to implement. The documentation specifically states:

[7] Related UvA Bugzilla ID 7561
[8] Related UvA Bugzilla ID 7564

> "Kurogo has the ability to support different languages and localities. This includes the translation of fixed on-screen values, error messages and other text strings. Currently Kurogo only includes translations for US English, however users can create their own translations and include them in their own site or submit it to Kurogo for inclusion in future versions." [14]

Investigating the framework showed that Kurogo does indeed support localization, however exclusively system-wide localization. It is possible to create `ini` translation files with localization strings; however, these override all localized texts for the entire system, with no possibility to only change the language for a specific user or session.

Thus, this feature was implemented for this project. This was done similarly to how the visitor can call the website with a `GET` variable to force a device type: `/?setdevice=compliant` will set the device type to `compliant` for that specific session. In order to honour Kurogo's conventions as much as possible, an extra flag was added: `setlanguage`. Kurogo uses ISO country and locality codes [9] such as `en_US` or `nl_NL` for language configuration. After implementation of this feature, UvApp now allows changing the system language *for that session* by requesting `/?setlanguage=nl_NL`. It was decided to only set the language for that specific session, because it would allow an implementation that required no persistent storage for user settings, something Kurogo does not support out-of-the-box. Making the setting persistent would require a far more extensive implementation, which was not worth the effort according to the stakeholders. Links were added in the UvApp menu for Dutch (`nl_NL`) and (U.S.) English (`en_US`). U.S. English was chosen as opposed to British English, because it was a default language file the platform provided.

4.3.3 Modules

This section will describe the various implemented UvApp modules. In addition to the implementation details, screenshots will be provided to give an idea of the visual representation.

Home

The Home module is the landing page. When a user navigates to UvApp, they will be met with either the screen shown on figure 4.2 for a compliant device (such as an Android or iOS smartphone) or the screen shown on figure 4.3 for a tablet device.

On a compliant device, the Home module is simply a menu, similar to how Android and iOS home screens are organized. The presentation scales with resolution. An app-wide search bar is integrated, which searches through every searchable module.

However, on a tablet device, the app has a permanent navigation bar on the left hand side. The search bar is integrated at the top, just below the logo (currently still showing the placeholder image with the text "Universitas"). The body of the page is an overview of several other modules. The modules displayed here are configurable in the administration console. It currently displays the News, Video, Calendar and Map modules.

Grades

The Grades module shows a student their grades, ordered by date in descending order. Every box is a grade, with the course name as header text and course date, ECTS and course catalogue ID as subtext. Underneath, the grade is displayed in large and red. Course names that are too long are displayed on several lines, rather than being truncated.

[9] See the following link for a list of supported language codes: `https://modolabs.jira.com/wiki/display/KDOC/List+of+Languages+Supported+by+Kurogo`

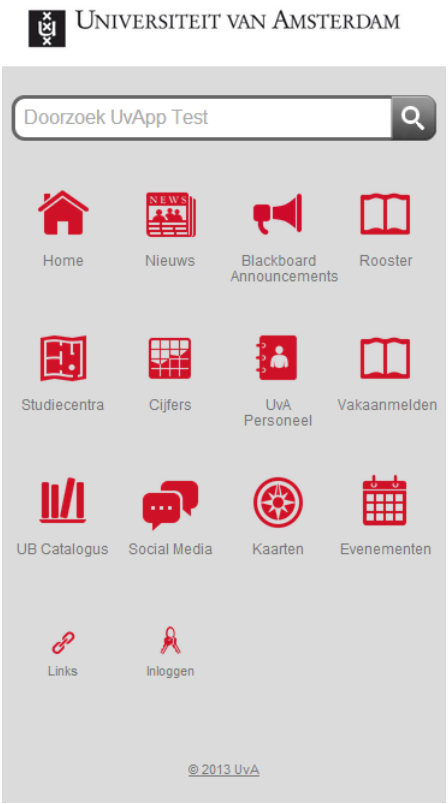

Figure 4.2: The Home module for a compliant device, such as an Android or iOS smartphone

Grades are obtained from a local dummy data file in JSON. However, the dummy data provided by the ICTS has an identical structure to a request from the actual production environment. In production, data would be obtained by means of a `GET` request to SIS.

Social

The Social module shows posts by the university's various social media accounts. Currently, the *UniversityofAmsterdam* Facebook account [10] and the FGw's *UvA_Student* Twitter account [11]. At the top of the page, there buttons that directly link to the university's social media accounts on the respective websites. Furthermore, an overview is given (ordered by date, descending) of all posts. Clicking on a tweet makes new buttons appear which allow the user to Retweet, Reply or Favorite.

This is an out-of-the-box Kurogo module which required only some INI configuration, in order to set up the various accounts.

UB Catalogue

The UB Catalogue module is a search box, as shown in figure 4.6. Upon entering a search text and pressing the `Search` button, the UB's mobile catalogue is queried and the results are shown in the app, as shown in figure 4.7.

UvA Staff

This module allows a logged in user to look up contact details for UvA staff. Figure 4.8 shows a

[10] `http://www.facebook/com/UniversityofAmsterdam`
[11] `http://www.twitter.com/UvA_Student`

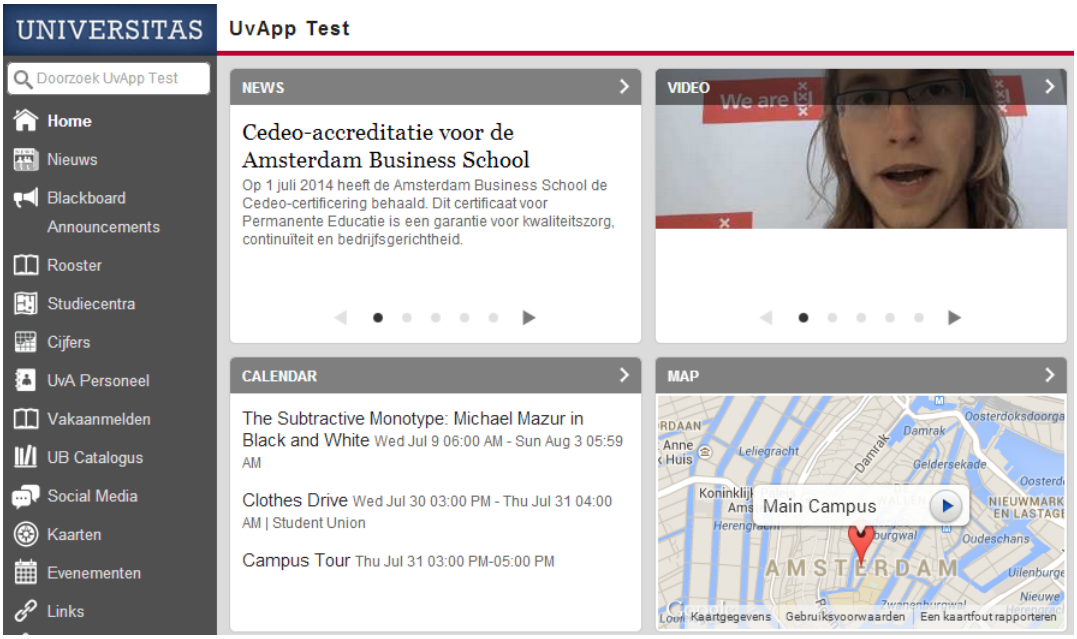

Figure 4.3: The Home module for a tablet device, such as an Android tablet or iPad

query for "Kuipers". On the left hand side is the list of results, on the right hand side the detail page. The "email" and "phone" boxes are clickable, which will open the users associated E-mail and Phone apps respectively, with the relevant information inserted.

Information for this module is retrieved from the UvA LDAP backend. However, it is only possible to search by last name, as full names are not entered in the LDAP backend (see also section 5.3.1). If the LDAP data was to be supplemented with first names, it would be possible to search by first name or full name.

Workstations

Interestingly, this module combines information from several data sources. First, the list of workstations and their amount of seats (free and total) are obtained from an XML feed. This feed is structured as follows:

```
<studiecentrabezetting TimeStamp="1-1-2014 0:00:00">
    <studiecentrum>
        <naam>SC-FNWI</naam>
        <werkplekken>199</werkplekken>
        <beschikbaar>150</werkplekken>
    </studiecentrum>
    ...
</studiecentrabezetting>
```

For every workstation, there is a codename (in this case `SC-FNWI`), the amount of seats and the free amount of seats. However, the codenames will not be understood by most students: `SC-AVZ, SC-FDR` and so forth. Therefore, in the configuration file for this module, there is a section called `studiecentra`. This section contains an amount of key-value pairs, with keys being the codenames and the values being the location names. However, these location names correspond to the names as found in the UvA's OpenAPI (currently just an internally used proof-of-concept).

Figure 4.4: The Grades module

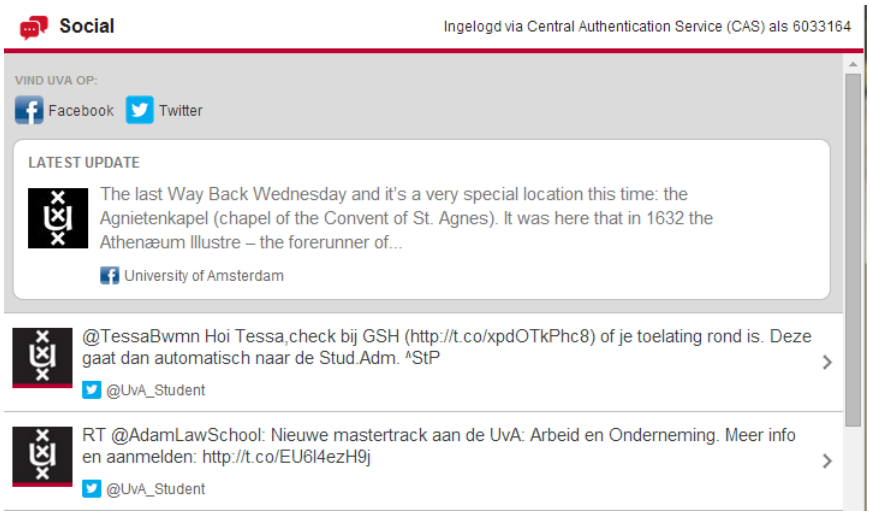

Figure 4.5: The Social module

Subsequently, more building data is obtained from this OpenAPI, such as the address. This building data is similarly entered as data for the Maps module, which in turn allows us to link to the Maps module with an address entered. As a result, `SC-FNWI` is translated to "Studiecentrum Science Park", which is a clickable link to the Maps module with the Science Park address entered and shown on the map.

In addition, opening hours are obtained from the various Google Calendar feeds which house this data (these feed addresses were obtained from the ICTS). In another section in the configuration file, called `openingstijden`, key-value pairs which link the codenames (`SC-FNWI`) to feeds, the opening hours are retrieved if available. This data is presented to the user as well.

In sum, by linking 3 data sources, the student can be provided with insightful information in one overview. Furthermore, they are able to simply click on a location and instantly see it on the Map. From there, a user can use the Maps module's "Get directions with Google Maps" link to quickly navigate to this location using GPS. Figure 4.9 shows the presentation of this module.

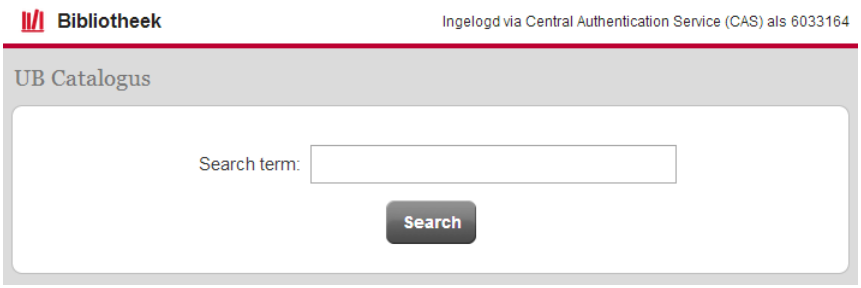

Figure 4.6: The UB Catalogue module

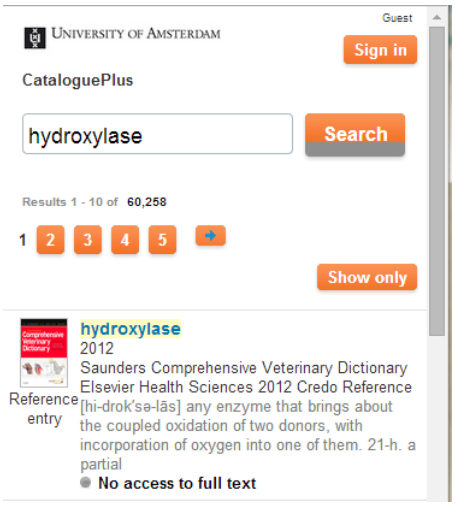

Figure 4.7: Results of searching the UB for "hydroxylase"

Maps

The Maps module is an out-of-the-box Kurogo module which presents the user with an interactive Google Maps embed. Locations can be selected from a list by clicking a "Browse" button, but can also be sought through by entering text in a search box. Figure 4.10 shows the presentation of this module. Locations are entered dynamically by means of a KML feed obtained from the UvA OpenAPI.

4.4 Development timeline

In this section, the timeline for the UvApp project will be outlined. The project officially started April 2014 and ended June 2014. However, a demo was given to the UvA's Mobile Expertise team in July. During this demo, the app's various features were showcased as part of a larger meeting regarding the future of mobile services for UvA students. For the rest of the project, a rough outline of the weekly activities are described below.

07/04/2014 Familiarized with the Kurogo platform and the work environment. Set up a Subversion repository, a Bugzilla group, a webserver for development purposes et cetera. A new Kurogo project was set up and the example Universitas app was studied to observe how the Kurogo platform works.

14/04/2014 Meeting with FGw. Decided on modules and features. Started development of first modules: Blackboard and Workstations ("studiecentra"). Workstations module still very

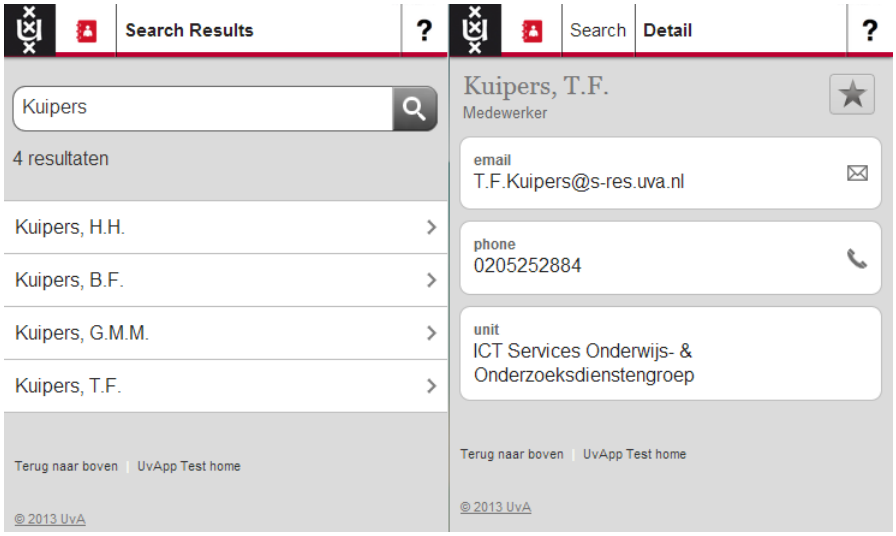

Figure 4.8: Looking up contact details for Tom Kuipers

Figure 4.9: Available workstations, their locations and opening hours

primitive, only showing workstation codenames and amount of seats.

21/04/2014 Connected Workstations module to locations by linking with OpenAPI. Also added opening hours, which are obtained from Google Cal feeds. Started rooster module.

05/05/2014 Spent significant time fixing the Maps module, as it would not work with Dutch localization due to seperator issues (replaced all dot seperators with commas, which - when entered into a JavaScript function - were seen as parameter delimiters). Moreover, linked Workstations to the Maps module and added OpenAPI buildings to the Maps module's location list. Started Grades module. Furthermore, cleaned up code and implemented some fixes for the Kurogo platform (see Contributions).

12/05/2014 Fixed People module (see Contributions, `LDAPPeopleRetriever` paragraph), adding initials. Added UB Catalogue search module. Configured and fixed Social module (see Contri-

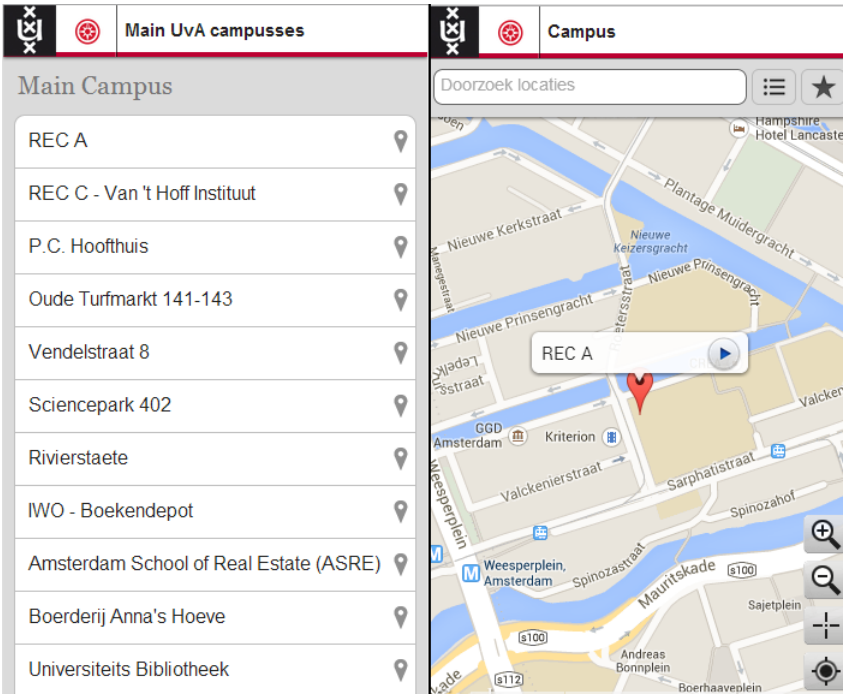

Figure 4.10: Left: Browse locations. Right: After clicking a location

butions, `FacebookDataRetriever` paragraph). Other minor fixes to the platform.

19/05/2014 More cleanup and aesthetic changes. Refinement of existent modules.

26/05/2014 Implemented Dutch localization feature.

02/06/2014 Administration console changes (see section 4.3.2). More module refinement.

09/06/2014 Translated Kurogo strings and out-of-the-box modules, creating a Dutch localization ini.

16/06/2014 Final refinements.

17/07/2014 Demo to the Mobile Expertise team.

CHAPTER 5

Results and Contributions

This chapter describes the results of the research that was done for this project. To reiterate, the main research question is whether, or to what extent, challenges in mobile software engineering can be overcome by use of mobile middleware. Furthermore, in order to answer this question, the following three secondary questions are noted:

- What are the typical challenges found in mobile software engineering?
- What are the trade-offs when using mobile middleware?
- What new challenges does using mobile middleware impose?

The results of the investigation, answers to these questions and contributions to the field made with this project are described in the coming sections.

5.1 Results: Frequency and extent of challenges in literature

First, the frequency and extent of the challenges mentioned in in (relatively recent [1]) literature, as experienced during this thesis project, are described. In order to do this, it is useful to reiterate and summarize the challenges mentioned.

- Fragmentation across platforms [39] [49] [51] [44]
- In-platform fragmentation [39] [51]
- Complexity of testing [39] [51]
- Lack of, or incomplete, automated testing tools [39]
- Mobile data is costly and unreliable, but storage capacity is limited too [39] [51] [49] [44] [48]
- Hard to keep up with frequent changes [39]
- Migrating code does not work properly due to fragmentation [39]
- Time, effort and budget are multiplied due to the different platforms [39]
- Emulators do not always work properly, lack functionality or are very slow [39]

[1] Since the year 2000. Due to the explosive growth of mobile technology, older literature was often no longer relevant.

- Mobile web applications can not use the full range of device features [39] [48]
- Limited screen space and hardware power [39] [51] [48]
- Security (related to wirelessly transmitted data, but also due to users not updating exploitable apps) [51] [17]

Other challenges are found on message boards and articles online. However, these are either Enterprise-only challenges (such as back-end integration with CRM/ERP systems or market research cost) or can be reduced to a challenge mentioned in the list above. It would appear that the above list is close to exhaustive. This also answers the first secondary research question.

Fragmentation (across platforms and within platform)

In section 3.1, this challenge is described extensively. To summarize, this challenge entails the great amount of different mobile devices, platforms and software versions that are being used. Furthermore, rather than becoming more unified, there is a trend of moving towards even more fragmentation [39].

Does mobile middleware work around this problem? Partially, inclining towards Yes. Mobile middleware such as Kurogo or PhoneGap are used to build web apps, with the possibility of generating a native wrapper in order to create a hybrid application. Modern web apps are written in HTML5 and JavaScript, which are being interpreted by the browser rather than the system. The major mobile browsers all support HTML5 relatively well (70%-90% of features, see section 3.1), with support increasing. HTML5 was designed with mobile in mind, making it a plausible alternative for native when it comes to building interactive apps.

By choosing to build a web app, the developer circumvents having to develop the app for the various platforms separately. One codebase, one web app, will suffice for all platforms. There is no need to take into account independent operating system version APIs, deprecation of old functions, forward compatibility and so forth. However, some constraints apply: not the full range of device functionality can be used with web, the performance does not match that of native (i.e. not usable for more graphically intense apps), and an internet connection is always required. These constraints are more extensively described in section 3.1.

Moreover, there are still the various screen sizes and densities to take into effect. Fragmented screen space is an issue that mobile middleware will not inherently circumvent. However, web design can be designed to scale with screen. Furthermore, mobile middleware tends to take into account the various screen sizes, and will automatically generate multiple presentations for the various types of screens (see also section 4.3.1).

During the creation of the UvApp, this theory was confirmed. Due to not needing any of the functionality that HTML5 does *not* provide, nor being hindered by the constraints of web technology, mobile middleware seemed an optimal choice for having to create the application only once, and not be hindered by the fragmentation challenge. Indeed, no regard was held for the different platforms or even screen sizes: the mobile middleware successfully managed it. Testing on various devices [2] confirmed that the middleware did indeed generate different presentations suitable for the device.

In sum, web technology inherently solves having to develop for multiple platforms natively. Moreover, mobile middleware solves the fragmentation in screen sizes and densities. Provided the constraints of web apps are not a problem, mobile middleware largely overcomes this problem.

[2] A Blackberry Bold 9700 running BB OS 5.0, an Android 4.x smartphone, an Android Honeycomb tablet, a second generation iPad, an iPhone 5s with iOS 6.x and 7.x, and a 25" 1080p monitor.

Testing

Extensively described in section 3.2, testing on mobile is lacking in several aspects. First and foremost, there are aspects on mobile devices such as low-power CPU, inconsistent and widely variable internet access, location services, battery life and so forth, that make testing significantly more complex as they are hard to reproduce. In addition, emulators are slow and do not reflect reality: there is no support for sensors, for instance. Furthermore, literature on automated testing and automated test suites are mostly limited to GUI testing, with little to no support for other classes of bugs. It must be noted that some aspects of testing are inherently just technically very difficult: how battery life is affected is a confluence of many factors. Debugging battery life (in the context of an application that does not have memory leaks and/or incorrectly uses the CPU) is very hard, and in practice mostly comes down to eliminating possible influences until the culprit is found.

Manual testing is prevalent over automation [39], but whether this is actually a problem in practice depends on context. One of the main advantages of automated testing is that it is time- and cost efficient [45]. Particularly when time and budget are scarce, for instance due to developing native apps for all the platforms (i.e. time and budget are multiplied, see several paragraphs below), this can be a large problem. However, when building a web application, only one codebase has to be tested. This, in itself, largely reduces the amount of code to be tested, making lack of automation less of a problem. Furthermore, since mobile web apps largely behave like regular web apps, the same test suites used for web applications can be used. While this still does not cover the inability to test sensor data, it does provide the developer with more mature and extensive testing tools.

To conclude, does mobile middleware solve the problem of incomplete testing tools and lack of automated testing? Again, partially. Due to the reasons described above (one codebase, mature testing tools), simply being a web app rather than a native app already solves part of the problem. However, it does not solve the lack of testing for mobile-specific scenarios mentioned in the first paragraph, such as how the app performs with inconsistent internet access or how it affects battery life. It is expected that as the mobile field matures, more advanced test tools will become available that will better resemble practice. For now, mobile middleware itself (i.e. other than being web, and moving into "web app testing" territory) does nothing for this challenge.

Mobile data is unreliable and costly

Section 3.3 describes the problem with mobile data: it is unreliable in availability and stability, varies in speed (from 56 Kbit/s GPRS to 1 Gbit/s from LTE Advanced) and tends to be expensive for consumers. This implies that, if available at all, network data transmission should be kept to a minimum. However, there is another concern that ties in directly with this one: storage capacity is limited. This means the device can not be loaded with a large amount of data (in advance, or in the presence of a broadband connection) either. Since large amounts of data can not be stored on the device in advance, nor can they be transmitted, the device should receive as little data as it can, while still being able to function.

What does mobile middleware do for this particular problem? Nothing. Being a web app, the app inherently relies on a functional internet connection. Moreover, the app transfers the entirety of the current view every page request. While the browser caches images, JavaScript and CSS files, the full HTML contents are sent for every page. Despite being only several kilobytes and often not significant enough to pose a problem in practice, this is a step back from native. With a native app, the server and client can exchange data in the most efficient formats possible, with the presentation being stored on the device in advance.

Hard to keep up with frequent changes

As described in section 3.1, not only are new devices released more quickly than old ones are discarded, but system updates are being rolled out in a high tempo. Often, these provide new standards, new APIs and deprecation of old ones. Developers note that it is difficult to keep up with these frequent changes, particularly because there is not one, but several platforms to keep up with.

Mobile middleware somewhat solves this issue. Once more, due to being web technology, the developer only has to keep up with HTML5 standard and implementation changes. These are far less frequent, with most of the standard already implemented in the major browsers. Moreover, there are various sites such as CanIUse [3] and HTML5Test [4] quickly showing what features can be safely used. Furthermore, the mobile middleware platform takes care of keeping up with part of the updates as well. To conclude, the developer still has to keep up with technology, albeit at a far slower pace and for the web platform only.

Migrating code does not work properly / Time, effort and budget are multiplied

Developers note that it is hard to migrate code to different platforms, due to different APIs, different requirements, different policies and so forth. As the different platforms have very different ways of implementing features such as notifications, code is hardly reusable [39]. These issues are, once more, inherent to native and do not apply to web, as code does not need to be ported in order to be able to run on other platforms.

Similarly, as code does not need to be ported (there is only one app), time, effort and budget are not multiplied per platform. Mobile middleware does indeed overcome these challenges, compared to writing native apps.

Emulators do not always work properly, lack functionality or are very slow

Emulators which do not correspond to how devices work in practice make testing more difficult. As a consequence, testing on emulators is limited and results of testing are less reliable. Please see paragraph “Testing” on the previous page for more information.

Mobile web applications can not use the full range of device features

Unfortunately, there is HTML5 functionality which is either not implemented by the web browsers or has only been implemented recently. Some HTML5 features, such as the `FileSystem API` (virtual filesystem for persistent storage) are indeed not supported by any of the devices as of this writing. However, mobile-specific features such as geolocation and accelerometer are currently available on all of the major mobile browsers.

However, it must be noted that mobile middleware can be the solution to a part of this problem. In generating a native wrapper for a hybrid application, the native wrapper can take the device features which the browser does not support by itself. Data can subsequently be communicated to the web app, acting as an intermediary. For instance, the FileSystem API is currently not supported by any of the mobile browsers. However, native apps can use the file system extensively. The web app could send data to the native wrapper and instruct it to persistently store it, and later ask the native wrapper to retrieve the data. In essence, the web app has now used the file system. As the middleware used in the UvApp project does not support generating a native wrapper without a commercial license, this could not be investigated further. It can be reasoned, however, that mobile middleware could indeed overcome this challenge.

[3] `http://www.caniuse.com`
[4] `http://www.html5test.com`

Limited screen space and hardware power

Limited screen space and hardware power are a challenge to software engineers because it forces them to rethink how to use these resources. On non-mobile devices such as PCs, processing power is often abundant and battery life due to high CPU usage is not an issue. However, on mobile devices, CPU usage directly means battery drainage. Thus, processing power should be used as sparingly as possible.

Moreover, while the most recent smartphone flagships have 1080p screens [7] [19] [11], screen sizes still generally range from 4" to 5" in diagonal [2] [3]. Less recent smartphones generally have a resolution of 320x480 [2] [3]. In contrast, 99% of the PCs that use web browsers have a screen resolution of 1024x768 [25], with approximately 34% of screens being considered "High Resolution" [26] (1360x768 through 2560x1440) as of January 2014. Furthermore, these PC resolutions tend to be displayed on external monitors, which are often no less than 15" in size. Not only is there a larger resolution to work with, the dots-per-inch (dpi) is also less dense. This causes for greater overview, and allows the screen to be filled with more elements.

Thus, developers will have to closely consider how they use screen space and processing power. Apps should be lightweight in resource usage and should have a clean and concise user interface. While these recommendations apply to traditional software, they apply even more to mobile.

The question is, can mobile middleware overcome this challenge? In theory, somewhat. While mobile middleware can not take away these hardware limitations, it can aid the developer in building mobile-friendly user interfaces. Likewise, it can aid the developer in building lightweight apps by using efficient and well-written libraries. Finally, as mobile middleware builds web apps, it can move part of the logic processing to a remote server (the web server). As a trade-off, the web app's HTML presentation is being parsed by the browser and the JavaScript interpreted by a JS engine, rather than as native code. Whether the trade-off is favorable depends on the complexity of the algorithms and the amount of data to be processed. In this thesis project, the Kurogo middleware definitely helped build suitable, clean UIs for the various screen sizes. It allowed the developer not to worry about the limited screen space, but rather what data was to be presented and the underlying logic.

5.2 Results: New challenges found

In this section, any new challenges that were found during the investigation and the creation of UvApp, are provided.

First, it is noteworthy that there were no new challenges found which are inherent to mobile itself. All of the challenges that became apparent during this project which were not related to the mobile middleware platform, were already documented in literature.

However, there were several challenges which were imposed by the use of mobile middleware. While it made the development process easier and quicker (as opposed to building a mobile web app without a mobile-optimized framework), it also presented some difficulties. Furthermore, committing to a specific middleware platform poses several risks.

One of these risks is that, rather than learning the architecture and codebase of a specific native platform, the developer is learning how to use a separate framework. While the codebase of a specific native platform will be updated and change over time, it will still be more or less the same: there is always something to fall back to. In case of using a middleware platform completely uncoupled from any of the platforms, there is the risk that the platform is dropped by the owners. Should this happen, there is nothing to fall back to: all the effort put into studying the framework has become worthless. All existing code that relies heavily on the framework will not be reusable. Furthermore, there is great dependence on the owners of the platform

in providing the developer with sufficient documentation. In addition, a lively community is desirable.

In case of Kurogo, it initially seemed the documentation was extensive and, due to the large amount of universities using the framework, had a good community. However, in practice, it became apparent documentation was often lacking. As a result, the Kurogo core files had to be studied carefully in order to find out how specific functionality was to be used. In other cases, functionality existed which was nowhere to be found in the documentation. Only after programming specific functionality it later became apparent that a similar feature already existed – with no mention of it anywhere. The documentation also lacks example code: there is none whatsoever. Any "example code" is found in the default modules. There is no documentation on the built-in libraries other than the code itself.

Moreover, the community did not appear to be lively at all, and the questions far outnumbered the answers. There was no support, or even presence, from Modo Labs themselves on the boards. Finally, some of the advertised features, such as generation of a native wrapper, were only available upon buying a commercial license. This only becomes apparent once these features are attempted to be used, and is not transparent on their website.

The issues mentioned above seem severe. However, they must be weighed against the high amount of default modules that come with the platform. The dynamic presentation for the various device types and screen sizes is another large benefit. In addition, there are data connectors for most standard solutions. It is clear that this framework aims to allow the user to quickly create a mobile-friendly app, particularly for universities, without investing too much effort or doing too much actual programming.

To conclude, the new challenges found are inherent to committing to a specific framework. The framework must be studied, which is a risky investment due to uncertainty of long term support. Whereas it is unlikely that the first parties will drop support for their own platforms, there is a higher degree of uncertainty regarding the third party mobile middleware. Furthermore, rather than having to study the documentation for all the platforms, the developer only has to study the documentation for the middleware. However, if the documentation is not sufficient, the developer is forced to invest more effort into studying the framework in a less efficient way (i.e. reading the code). Furthermore, it might later become apparent that some functionality is only available under certain circumstances, such as buying commercial licenses. In sum, the developer moves from several platforms, to becoming dependent on one - external - platform.

5.3 Contributions

In this section, the contributions made to the field and/or the host organisation are described. Contributions to the field are fixes and additions for the mobile middleware platform Kurogo and this overview of challenges in the context of using middleware.

5.3.1 Contributions to the field and the Kurogo platform

This thesis provides a conceptual overview of challenges in mobile software engineering found in literature (see section 5.1), as well as the extent of their applicability in the context of using mobile middleware.

Furthermore, in the process of developing UvApp, various bugs were discovered in Kurogo. In addition, it was found that some assumptions were made (see paragraph PeopleWebModule) which did not apply in the UvApp context. In addition to applying the fixes locally, changes that seemed applicable to a larger audience were proposed to the Kurogo team. Kurogo's community

version is hosted on Github [5] and it seems that pull requests with fixes are encouraged by Modo Labs. In order to contribute to the middleware that made this project possible, these fixes were submitted by pull request and/or e-mail to the Kurogo team. The following fixes were submitted:

LDAPPeopleRetriever [6]

The "People" module, linked to as "UvA Personeel", allows a user to look up UvA staff contact data in order to email or phone them, or find other relevant data. This data is retrieved from an LDAP server which the university uses widely as its authentication and authorization back-end. This LDAP data is fairly complete, in that it also contains staff email adresses, surnames and initials. A typical response, searching for "Kuipers" for instance, would return something like:

```
LDAPPerson Object
       (
           [dn:protected] => uid=xxxxxx, ou=Medewerkers, o=Universiteit van
Amsterdam, c=Nl
           [fieldMap:protected] => Array
               (

                   [fullname] =>
                   [firstname] => givenname
                   [lastname] => sn
               )

           [attributes:protected] => Array
               (
                   [sn] => Array
                       (
                           [0] => Kuipers
                       )

                   [initials] => Array
                       (
                           [0] => T.F.
                       )

               )
       )
```

Kurugo's results list uses the 'fullname' field to display names. However, in absence of this field, it shows firstname + lastname. firstname is defined as 'givenname', which is a non-existant attribute in the LDAP object. Thus, it would only show surnames in the result list. Searching for Kuipers would net one 4 results, all simply displaying 'Kuipers', indistinguishable from one another until the details page would be opened. In order to work around this, there were two possible solutions: you either fill in the field that it expects ('fullname'), which required changes to the existing enrolled LDAP server used by the UvA (not viable), or changing the Kurogo core. The latter option was chosen: a few extra lines of code were added that initially try to use the 'fullname' field, but in its absence, fall back to concatenating the initials to the surname to obtain something like `Kuipers, T.F`. This is not something inherently wrong with Kurogo. However,

[5] `https://github.com/modolabs/Kurogo-Mobile-Web`
[6] Related UvA Bugzilla ID 7499

the original code assumes that the "fullname" field always contains data, which in practice does not always seem to be the case (for the UvA at least, it was not). Moreover, since the fix did not break any existing structures, it was submitted to the Kurugo team by e-mail.

FacebookDataRetriever [7]

The social media module aims to inform the user with the latest Twitter and Facebook posts from UvA accounts. The Twitter integration was seamless - through the administration panel, or via ini configuration, OAuth account credentials were entered (this is required as per the Twitter API), and any Twitter account's feed can be shown on the page. However, for Facebook, this integration was not quite as seamless. Despite having generated an API key, OAuth keys and so forth, an error would be displayed regarding a non-existant index in an array, and the module would not work. This is an issue mentioned at least once more on the Kurogo community boards [8], however not acted upon. Perhaps it is contextual and only occurs in certain cases. However, for us, the module would not work out-of-box. Index `count` was missing from the object, which after some playing with the Facebook API and its JSON responses, appeared to be an index that indicates the amount of likes per post. This index was either removed in a recent update, or not always present; the cause is unknown. It is intuitive to think that the index might not exist when a post does not have any "Likes", however, this is not the case: the index was simply never present in any response we obtained. Therefore, a patch was written for the `FacebookDataRetriever` library. In order not to break anything (in case the index somehow returned to the response), the code still checks if the index exists. If it does not, it calculates the likes by counting the amount of sub-objects in the 'likes' index, and returns that. This code was also proposed to the Kurogo team by means of a Github pull request [9].

PeopleWebModule [10]

The PeopleWebModule is the controller for the People module, which retrieves model data from some DataRetriever, parses it for viewing, and then passes data to the template. However, it was here that it first became apparent that Kurogo is indeed a platform maintained by an American company. Despite being used internationally, there are still some assumptions which do not make sense outside the United States. We encountered one of these when we were looking up a phone number in the People module, and found that for some reason every phone number was prepended with a `+1`. Despite having entered the default area code as `020` (Amsterdam), it still prepended a `+1` with unknown origin. After some searching, it was found that this prepend was hardcoded into the PeopleWebModule: +1 is the country code for North America, and upon finding a phone number missing this country code, it would prepend it. A notification was sent to the Kurogo team.

5.3.2 Contributions to the host organization

During this project, several contributions were made to the host organization.

First and foremost, the delivery of a proof-of-concept one-stop mobile web portal for students: UvApp. In a July 2014 demo to the Mobile Expertise Team, this app was well received and interest was expressed. As opposed to outsourcing building an expensive app, it was shown that a single student could make a competitive app which catered to students' needs, on a low budget in a 3-month timeframe.

[7] Related UvA Bugzilla ID 7565
[8] `https://groups.google.com/forum/#!topic/kurogo-dev/y3NeFVlf6Dc`
[9] `https://github.com/modolabs/Kurogo-Mobile-Web/pull/73`
[10] Related UvA Bugzilla ID 7500

Furthermore, the author was asked to provide feedback on the ICTS internship environment, and to assist in spotting possible new talent for recruitment. The author agreed to both. Finally, several suggestions were made to improve existing services, which were well received.

CHAPTER 6

Conclusion

It has become apparent that mobile devices have become an important part of many lives. Being a relatively new and fast-growing sector, there are still many challenges that face the development of functionality for these devices. These challenges, described extensively in section 3, are largely caused due to the lack of uniformity in devices. The many different software platforms and hardware compositions have not made mobile software engineering an easier task. Other challenges arise from the mobile technology itself and the various factors that in practice influence the device's behavior, but are hard to reproduce.

To recapitulate, in this thesis, the following research questions were asked and answered:

- **What are the typical challenges found in mobile software engineering?** Fragmentation caross and within platforms, complexity of testing, lack of (or incomplete) automated testing tools, costly and unreliable mobile data, limited storage capacity, hard to keep up with frequent changes, difficult to migrate code, multiplied time, effort and budget, emulators are incomplete and do not work as intended, mobile web can't use the full range of device features, limited screen space and hardware power and numerous security challenges. See also sections 3 and 5.1.

- **What are the trade-offs when using mobile middleware?** and **What new challenges does using mobile middleware impose?** Mobile middleware is a developer tool that can be used to write multi-platform code. This solves several of the challenges that mobile software engineers face, but is unable to solve others. Moreover, it actually imposes other potential problems. It is clear that mobile middleware can not overcome every challenge, but that is not to say it is not useful either. In fact, it can aid the developer in quickly building mobile-friendly web apps, without worrying too much about the "mobile" part. However, this is also the inherent limitation: it builds web apps, which do not really feel like a native app and can not match its performance or extensive featureset. There is still device functionality which is off-limits for web apps, and there is other functionality which works, but will never match the user experience of native.

  This is also where the trade-off lies in using mobile middleware. If the purpose of the developer is to build an information app that interacts with online services, does not require a lot of processing power or intense graphical operations, then the trade-off is worth it. A little user experience and device functionality is traded for cross-platform accessibility, ease of development, cost- and time efficiency. However, if the purpose of the app is to give a true "native app" feel or needs more of the device's capabilities, then the native choice is still the only option.

Does mobile middleware help overcome the challenges in mobile software engineering? Indeed, it helps - it does not allow a developer to completely overcome them. For some challenges, it can be a great aid. For others, not at all. Please refer to section 5 for more detailed information per challenge.

6.1 Acknowledgements

The author would like to thank Tom F. Kuipers, Alan M. Berg and the University of Amsterdam's ICTS department for providing the opportunity to do this project, as well as a pleasant work environment.

Bibliography

[1] Android - Phones and Tablets. http://www.android.com/phones-and-tablets/.

[2] Android Developer site - Supporting Multiple Screens. http://developer.android.com/guide/practices/screens_support.html.

[3] Android Developers - Dashboards. https://developer.android.com/about/dashboards/index.html.

[4] Android Developers - Supporting Different Platform Versions. https://developer.android.com/training/basics/supporting-devices/platforms.html.

[5] Appium - Mobile App Automation Made Awesome. http://appium.io/.

[6] Apple - App Store Distribution. https://developer.apple.com/support/appstore/.

[7] Apple - iPhone 5 Technical Specifications. http://support.apple.com/kb/sp655.

[8] BGR News - "Despite iPhone 6 hype, Android continues to dominate iOS market share". http://bgr.com/2014/07/01/android-market-share-2014/.

[9] CanIUse.com - HTML5 support charts.

[10] Experitest - Mobile Testing & Monitoring Tools. http://experitest.com/.

[11] HTC US - One X Specs. http://www.htc.com/us/smartphones/htc-one-x/.

[12] jQuery Mobile - A Touch-Optimized Web Framework. http://jquerymobile.com/.

[13] Kurogo Documentation - Administration Console. https://modolabs.jira.com/wiki/display/KDOC/Administration+Console.

[14] Kurogo Documentation - Localization - https://modolabs.jira.com/wiki/display/KDOC/Localization.

[15] Kurogo Examples - http://kurogo.org/examples/list.php.

[16] Kurogo Technology - http://kurogo.org/technology/.

[17] Mobile Development Challenges - 2011 Oracle Week Israel presentation. Slides at http://www.slideshare.net/levynir/mobile-development-challenges-10288414.

[18] Modo Labs - Kurogo - open-source Mobile Optimized Middleware.

[19] Nexus 5 Technical Specifications - Google. http://www.google.com/nexus/5/.

[20] OpenSignal - Android Fragmentation Report July 2013. http://opensignal.com/reports/fragmentation-2013/.

[21] PhoneGap. http://www.phonegap.com.

[22] Samsung Galaxy S4 Technical Specification. http://www.samsung.com/latin_en/consumer/mobile-phones/mobile-phones/smartphone/GT-I9500ZKLTPA-spec.

[23] Techcrunch - "Windows Phones Market Share In The United States Isnt Growing". - http://techcrunch.com/2014/07/03/windows-phones-market-share-in-the-united-states-isnt-growing/.

[24] Techopedia. Definition of Mobile Middleware. http://www.techopedia.com/definition/24433/mobile-middleware.

[25] W3schools - Browser Display Statistics. http://www.w3schools.com/browsers/browsers_display.asp.

[26] W3schools - High Screen Resolution Statistics. http://www.w3schools.com/browsers/browsers_resolution_higher.asp.

[27] Wikipedia - Comparison of Android Devices. http://en.wikipedia.org/wiki/Comparison_of_Android_devices.

[28] WorkLight Webinar Series - Native, Web or Hybrid Mobile App Development? http://www.scribd.com/doc/50805466/Native-Web-or-Hybrid-Mobile-App-Development.

[29] ZDNet - Storage in 2014: An overview. http://www.zdnet.com/storage-in-2014-an-overview-7000024712/.

[30] D. Amalfitano, AR. Fasolino, and P. Tramontana. A GUI Crawling-Based Technique for Android Mobile Application Testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 252–261, March 2011.

[31] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. Using GUI Ripping for Automated Testing of Android Applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 258–261, New York, NY, USA, 2012. ACM.

[32] Gary Anthes. HTML5 leads a web revolution. *Commun. ACM*, 55(7):16–17, July 2012.

[33] M. Becher, F.C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf. Mobile Security Catching Up? Revealing the Nuts and Bolts of the Security of Mobile Devices. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 96–111, May 2011.

[34] Technews Daily. Smartphones Outsold PCs for the First Time in 2010. http://www.technewsdaily.com/2028-smartphones-outsold-pcs-for-the-first-time-in-2010.html. 2011.

[35] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[36] Cuixiong Hu and Iulian Neamtiu. Automating GUI Testing for Android Applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 77–83, New York, NY, USA, 2011. ACM.

[37] Edward Iglesias and Wittawat Meesangnil. Mobile website development: From site to app. *Bulletin of the American Society for Information Science and Technology*, 38(1):18–23, 2011.

[38] Business Insider. The Smartphone Market Is Now Bigger Than The PC Market. http://www.businessinsider.com/smartphone-bigger-than-pc-market-2011-2. 2011.

[39] M.E. Joorabchi, A. Mesbah, and P. Kruchten. Real Challenges in Mobile App Development. In *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, pages 15–24, Oct 2013.

[40] Martin Kropp and Pamela Morales. Automated GUI testing on the android platform. *on Testing Software and Systems: Short Papers*, page 67, 2010.

[41] Neal Leavitt. Mobile Security: Finally a Serious Problem? *Computer*, 44(6):11–14, June 2011.

[42] Riyadh Mahmood, Naeem Esfahani, Thabet Kacem, Nariman Mirzaei, Sam Malek, and Angelos Stavrou. A whitebox approach for automated security testing of Android applications on the cloud. In *Automation of Software Test (AST), 2012 7th International Workshop on*, pages 22–28. IEEE, 2012.

[43] Amiya Kumar Maji, Kangli Hao, Salmin Sultana, and Saurabh Bagchi. Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian. In *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ISSRE '10, pages 249–258, Washington, DC, USA, 2010. IEEE Computer Society.

[44] Euler Horta Marinho and Rodolfo Ferreira Resende. Quality Factors in Development Best Practices for Mobile Applications. In *Proceedings of the 12th International Conference on Computational Science and Its Applications - Volume Part IV*, ICCSA'12, pages 632–645, Berlin, Heidelberg, 2012. Springer-Verlag.

[45] H. Muccini, A Di Francesco, and P. Esposito. Software testing of mobile applications: Challenges and future research directions. In *Automation of Software Test (AST), 2012 7th International Workshop on*, pages 29–35, June 2012.

[46] Jon Oberheide and Farnam Jahanian. When Mobile is Harder Than Fixed (and Vice Versa): Demystifying Security Challenges in Mobile Environments. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10, pages 43–48, New York, NY, USA, 2010. ACM.

[47] PewInternet. Mobile Technology Fact Sheet 2014, http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/. 2014.

[48] D. Sin, E. Lawson, and K. Kannoorpatti. Mobile Web Apps - The Non-programmer's Alternative to Native Applications. In *Human System Interactions (HSI), 2012 5th International Conference on*, pages 8–15, June 2012.

[49] P. Smutny. Mobile development tools and cross-platform solutions. In *Carpathian Control Conference (ICCC), 2012 13th International*, pages 653–656, May 2012.

[50] T. Takala, M. Katara, and J. Harty. Experiences of System-Level Model-Based GUI Testing of an Android Application. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 377–386, March 2011.

[51] Anthony I. Wasserman. Software Engineering Issues for Mobile Application Development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 397–400, New York, NY, USA, 2010. ACM.

[52] Shun Han Rebekah Wong. Which platform do our users prefer: website or mobile app? *Reference Services Review*, 40(1):103–115, 2012.

[53] Wei Yang, MukulR. Prasad, and Tao Xie. A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications. In Vittorio Cortellessa and Dniel Varr, editors, *Fundamental Approaches to Software Engineering*, volume 7793 of *Lecture Notes in Computer Science*, pages 250–265. Springer Berlin Heidelberg, 2013.

[54] Cong Zheng, Shixiong Zhu, Shuaifu Dai, Guofei Gu, Xiaorui Gong, Xinhui Han, and Wei Zou. Smartdroid: an automatic system for revealing UI-based trigger conditions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pages 93–104. ACM, 2012.