

Master's Thesis

# A generic approach towards smart self-rendering mobile user interfaces

Peter Treitler

Institute for Information Systems and Computer Media,  
Graz University of Technology



Supervisor: Assoc. Prof. Andreas Holzinger, PhD, MSc, MPh, BEng, CEng, DipEd,  
MBCS

Graz, March 12, 2012

This page intentionally left blank

Masterarbeit

(Diese Arbeit ist in englischer Sprache verfasst)

# Ein generischer Ansatz zur Erstellung intelligenter mobiler Benutzeroberflächen

Peter Treitler

Institut für Informationssysteme und Computer Medien,  
TU Graz



Betreuer: Univ.-Doz. Ing. Mag.rer.nat. Mag.phil. Dr. Andreas Holzinger

Graz, 12. März 2012

This page intentionally left blank

# Abstract

Today, developers of mobile applications face a major challenge: Screen sizes of different devices are varying to a large extent – along with different aspect ratios. Since the Android platform becomes more and more available on the market, this issue becomes more important.

Different screen sizes along with the increasing complexity of mobile tasks create a serious obstacle to usability and justify research to overcome these obstacles (Chae and Kim (2004)). One way to circumvent the obstacle is to organize an information structure with efficient depth/breadth trade-offs.

The goal of this thesis is the experimental research on both platform-specific and universal mobile user interfaces and their scalability and interoperability across platforms and screen sizes. A possible approach is to adapt the Model View Controller (MVC) design pattern, which has been developed more than 30 years ago, however, has big potential for future web applications, especially in terms of usability issues (Holzinger et al. (2010)).

On the basis of a systematic analysis of related work, this thesis will investigate possibilities which the currently most common mobile platform application programming interfaces (APIs) offer to the developers in order to ensure full scalability of their user interfaces. It will systematically analyze existing mobile applications and the solutions on how they deal with these problems and also perform research in order to find new approaches. The interoperability of different systems, including HTML5, Java and .NET will also be in the focus of this thesis.

**Keywords**

Mobile computing, usability engineering, user interfaces, methodologies

**ÖSTAT classification**

1108 Informatics

1138 Information Systems

1157 Usability Research

1161 Human-Computer Interaction (HCI)

**ACM classification**

D.2, H.5

H. Information Systems

H.5 INFORMATION INTERFACES AND PRESENTATION (I.7)

H.5.2 User Interfaces (D.2.2, H.1.2, I.3.6)

Subjects: Evaluation/methodology, Graphical user interfaces (GUI), Screen design

This page intentionally left blank

## Kurzfassung

Entwickler von Software für mobile Geräte sehen sich heute mit vielen neuen Herausforderungen konfrontiert. Eine davon besteht darin, dass aktuelle mobile Geräte sehr unterschiedliche Bildschirmgrößen sowie Seitenverhältnisse haben. Durch den zunehmenden Erfolg der Android Plattform gewinnt dieses Problem zusätzlich an Bedeutung. Es ist keine einfache Aufgabe, Anwendungen zu entwickeln die auf einer Vielzahl solcher Geräte lauffähig und benutzerfreundlich bedienbar sind.

Die verschiedenen Bildschirmgrößen stellen zusammen mit den immer komplexer werdenden Aufgaben, die auf mobilen Geräten erledigt werden ein großes Hindernis dar, welches die Notwendigkeit von Forschung in diesem Bereich deutlich macht (Chae and Kim (2004)).

Eine Möglichkeit, diese Hindernisse zu umgehen ist die Verwaltung von Informationsstrukturen mit effizienten Kompromissen zwischen Tiefe und Breite.

Das Ziel dieser Arbeit ist die experimentelle Erforschung von plattformspezifischen und universellen mobilen Benutzeroberflächen sowie deren Skalierbarkeit und Interoperabilität. Eine mögliche Herangehensweise ist eine Adaption des Model View Controller (MVC) Entwurfsmusters, welches zwar schon vor über 30 Jahren erstmals entwickelt wurde, jedoch auch für zukünftige Applikationen großes Potenzial hinsichtlich Usability-Problemen besitzt (Holzinger et al. (2010)).

Auf Basis systematischer Analyse themenbezogener Arbeiten wird diese Arbeit die Möglichkeiten untersuchen, die die aktuell am weitesten verbreiteten mobilen Plattformen sowie deren Entwicklungsumgebungen (APIs) den Entwicklern bieten, um die Skalierung von Benutzeroberflächen zu ermöglichen.

Die Arbeit analysiert systematisch existierende mobile Anwendungen und wie diese mit den genannten Problemen umgehen. Auch weitere Forschung nach neuen Herangehensweisen wird angestellt. Ein weiterer Schwerpunkt dieser Arbeit liegt auf der Interoperabilität verschiedener Systeme, wie beispielsweise HTML5, .NET und Java.



### **Schlüsselwörter**

Usability engineering, mobile Geräte, Benutzeroberflächen, Methodologien

### **ÖSTAT Klassifikation**

1108 Informatics

1138 Information Systems

1157 Usability Research

1161 Human-Computer Interaction (HCI)

### **ACM Klassifikation**

D.2, H.5, J.1

H. Information Systems

H.5 INFORMATION INTERFACES AND PRESENTATION (I.7)

H.5.2 User Interfaces (D.2.2, H.1.2, I.3.6)

Subjects: Evaluation/methodology, Graphical user interfaces (GUI), Screen design

This page intentionally left blank

## STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, March 12, 2012

---

Peter Treitler

This page intentionally left blank

## Acknowledgements

I would like to thank Boom Software AG for the enjoyable and insightful cooperation throughout the project and for giving students the chance to do projects in cooperation with the company. I especially want to thank Joachim Schnedlitz, CEO of Boom Software AG, for making the project possible and Roman Bobik, our project manager and spokesperson at Boom Software, for the excellent and supportive cooperation. I would also like to thank Michael Geier for the good teamwork on the project.

I would like to thank Wolfgang Slany for giving me the chance to work on the Catroid project<sup>1</sup>. The insights gained throughout this project have been an important motivation for writing this thesis.

Finally, I would like to thank my supervisor, Andreas Holzinger, for his support and guidance during my work and for giving me the chance to work on the usability project with Boom Software AG.

Peter Treitler, BSc.  
Graz, March 12, 2012

---

<sup>1</sup><http://www.catroid.org>

This page intentionally left blank

# Table of Contents

<b>1</b>	<b>Introduction and Motivation for Research</b>	<b>17</b>
1.1	Mobile devices . . . . .	17
1.2	Mobile usability . . . . .	18
<b>2</b>	<b>Theoretical Background</b>	<b>21</b>
2.1	The relevance of mobile devices . . . . .	21
2.2	The Model-View-Controller (MVC) Pattern . . . . .	22
2.3	Screen size as a limiting factor . . . . .	23
2.4	Usability . . . . .	26
2.5	Usability evaluation methods . . . . .	27
2.5.1	Usability inspection methods . . . . .	28
2.5.2	Usability test methods . . . . .	29
<b>3</b>	<b>Related Work</b>	<b>31</b>
3.1	Definitions . . . . .	31
3.2	Mobile platforms . . . . .	32
3.3	Android . . . . .	32
3.3.1	Available devices . . . . .	32
3.3.2	Supporting different screen sizes and densities . . . . .	36
3.3.3	Scaling images . . . . .	37
3.3.4	Nine-Patches . . . . .	37
3.3.5	The Back Stack . . . . .	38

3.3.6	Fragments . . . . .	39
3.3.7	Android Design . . . . .	40
3.4	Windows Phone . . . . .	42
3.4.1	Screen sizes . . . . .	42
3.4.2	Defining layouts . . . . .	43
3.4.3	Supporting different screen sizes . . . . .	43
3.5	iOS . . . . .	44
3.5.1	Screen sizes . . . . .	45
3.5.2	Defining layouts . . . . .	46
3.5.3	Supporting different screen sizes . . . . .	48
3.5.4	Apple’s Human Interface Guidelines . . . . .	48
3.6	HTML and CSS . . . . .	49
3.6.1	Defining layouts . . . . .	50
3.6.2	Touch input . . . . .	50
3.7	Multiplatform development . . . . .	51
3.7.1	HTML5 versus native apps . . . . .	52
3.7.2	Multiplatform frameworks . . . . .	53
<b>4</b>	<b>Materials and Methods</b>	<b>59</b>
4.1	Introduction . . . . .	59
4.2	About Boom Software AG . . . . .	59
4.3	The Boom BORA framework . . . . .	60
4.4	The Boom Mobile app . . . . .	62
4.4.1	First prototype . . . . .	63
4.4.2	Criticism of the prototype . . . . .	65
4.4.3	Improved version . . . . .	67
4.4.4	Scalability . . . . .	68
4.4.5	Future development . . . . .	69
4.5	Usability evaluation of desktop applications . . . . .	71



4.5.1	Heuristic Evaluation . . . . .	71
4.5.2	Thinking Aloud Tests . . . . .	71
4.6	Usability guidelines . . . . .	79
<b>5</b>	<b>Results</b>	<b>81</b>
5.1	Usability evaluation results . . . . .	81
5.1.1	Heuristic evaluation . . . . .	81
5.1.2	Thinking aloud test . . . . .	83
5.2	Usability guidelines . . . . .	86
5.2.1	General UI design guidelines . . . . .	87
5.2.2	Framework-specific guidelines . . . . .	89
<b>6</b>	<b>Discussion and Lessons Learned</b>	<b>91</b>
6.1	Mobile UI design . . . . .	91
6.1.1	Choosing a method for app development . . . . .	92
6.2	Usability inspection and testing . . . . .	93
6.3	Usability guidelines . . . . .	95
6.4	The Boom Mobile app . . . . .	96
6.5	Scalability of Boom Software’s UIs . . . . .	96
<b>7</b>	<b>Conclusions</b>	<b>99</b>
<b>8</b>	<b>Future Work</b>	<b>101</b>
	<b>List of Figures</b>	<b>103</b>
	<b>List of Tables</b>	<b>105</b>
	<b>References</b>	<b>107</b>



# 1. Introduction and Motivation for Research

## 1.1 Mobile devices

Today, mobile devices are gaining more and more importance. Smartphones - mobile phones with greater computing power than ordinary mobile phones and internet connectivity - are a common sight nowadays in developed countries. What started as a concept product by IBM in 1992 was mostly a niche product for a long time until the release of the Apple iPhone in 2007, which started bringing smartphones to the mainstream. One year later Google and the Open Handset Alliance released their Android platform, which gained a lot of popularity in the following years. As of January 2012, there was a total of more than 300 million Android devices shipped worldwide, with more than 800,000 new ones being activated every day<sup>1</sup>.

Tablet computers are another type of mobile device becoming increasingly popular. Tablet PCs with the Windows operating system have existed for a while. The approach here was to create small-sized laptop PCs with touchscreens. Recently, smartphone manufacturers have begun to expand into the tablet computer market as well. The most common devices are Apple's iPad and various Android tablet computers.

Netbooks represent a third category of mobile devices. These are very small, lightweight laptop computers, which are often significantly less expensive than reg-

---

<sup>1</sup><http://googlemobile.blogspot.com/2012/02/androidmobile-world-congress-its-all.html>, retrieved Feb. 27, 2012

ular laptop computers as well. A wide variety of netbooks exists today. Operating systems commonly found on netbooks include Windows XP, Windows 7, Android and various Linux distributions.

Regular laptop computers are not inside the main focus of this thesis. While they are portable devices, they can be more easily compared to desktop computers than to the other devices mentioned in this section and some of the problems discussed don't apply to them. As pointed out by Schmidt (2000), the context of use is also different for laptops. Despite being called mobile devices, laptops are mostly used in a stationary way, i.e. carried to work, placed on a desk and used there. Smartphones and tablets in contrast are more commonly used in a truly mobile way - while riding the bus or train, while taking a walk in the park or wherever the user chooses to use them. These different scenarios of use imply different user tasks, which in turn should be considered when designing user interfaces.

## 1.2 Mobile usability

With so many different types of mobile devices being available and becoming more common, more and more complex applications become available for them. Keeping such complex applications easily usable on small devices such as smartphones can be a challenge. A small screen size, touchscreen interaction and the lack of a real keyboard are some of the factors that require consideration.

In addition to that, mobile devices come in a variety of different screen sizes. Small smartphones have a display diagonal of 2 inches while some tablet computers' displays measure 10 inches or more. This especially holds true when designing for multiple platforms, but the problem also arises with different devices of the same platform. Android smartphones and tablet computers cover the whole range mentioned and the difference between Apple's iOS devices is significant as well: The iPhone (4th generation) has a 3.5 inch display while the iPad's (2nd generation) display measures 9.5 inches.

It is a major challenge for developers to deliver software that offers a good user experience on different screen sizes without having to create two completely separate user interfaces.

The goal of this thesis is therefore the research on both native and cross-platform universal user interfaces. The Model View Controller (MVC) design pattern, which was first described more than 30 years ago by Reenskaug<sup>2</sup>, is a good approach towards creating scalable user interfaces. The separation of concerns introduced by the MVC pattern (see section 2.2) means that only the view must be changed while the rest of the applications code base can remain untouched. The MVC pattern therefore has big potential for future mobile apps and web applications especially in terms of usability issues (Holzinger et al. (2010)).

On the basis of systematic analysis of related work, this thesis will investigate possibilities which the currently most common mobile platforms' application programming interfaces (APIs) offer to the developers in order to ensure full scalability of their user interfaces. It will analyze platform standards and how existing mobile applications deal with the problems mentioned and attempt to find new approaches. Also the interoperability of different systems and the design of cross-platform user interfaces (for instance through HTML5) will be in the focus of this thesis.

---

<sup>2</sup><http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>, retrieved March 9, 2012



## 2. Theoretical Background

### 2.1 The relevance of mobile devices

In February 2012, IT market analysts Canalys released their estimated shipment figures of desktop PCs and mobile devices for the year 2011<sup>1</sup>. As shown in figure 2.1, smartphones have overtaken PCs (which in this figure include tablets) significantly, with 487.7 million smartphones sold in 2011 versus 414.6 million PCs. According to Canalys' figures, tablets (called Pads in the figure) have the highest growth rate with 274.2%, which places them ahead of netbooks in total shipments (63.2 million versus 29.4 million).

<b>Worldwide smart phone and client PC shipments</b>				
<b>Shipments and growth rates by category, Q4 2011 and full year 2011</b>				
Category	Q4 2011 shipments (millions)	Growth Q4'11/Q4'10	Full year 2011 shipments (millions)	Growth 2011/2010
Smart phones	158.5	56.6%	487.7	62.7%
Total client PCs	120.2	16.3%	414.6	14.8%
- Pads	26.5	186.2%	63.2	274.2%
- Netbooks	6.7	-32.4%	29.4	-25.3%
- Notebooks	57.9	7.3%	209.6	7.5%
- Desktops	29.1	-3.6%	112.4	2.3%

Source: Canalys estimates © Canalys 2012

**Figure 2.1:** Smartphone and PC shipments for 2011, as estimated by Canalys<sup>1</sup>.

When breaking the smartphone shipments down by platform (see figure 2.2), Android has the largest share with 48.8%, with iOS (19.1%) and Symbian (16.4%) on the second and third places.

<sup>1</sup><http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011>, retrieved Feb. 7, 2012)

Worldwide smart phone market Shipments by platform, Q4 2011				Worldwide smart phone market Shipments by platform, full year 2011			
Platform	Q4 2011 shipments (millions)	Share (%)	Growth Q4'11/Q4'10	Platform	Full year 2011 shipments	Share (%)	Growth Q4'11/Q4'10
Total	158.5	100.0%	56.6%	Total	487.7	100.0%	62.7%
Android	81.9	51.6%	148.7%	Android	237.8	48.8%	244.1%
iOS	37.0	23.4%	128.1%	iOS	93.1	19.1%	96.0%
Symbian	18.3	11.6%	-40.9%	Symbian	80.1	16.4%	-29.1%
BlackBerry	13.2	8.3%	-9.7%	BlackBerry	51.4	10.5%	5.0%
bada	3.8	2.4%	39.1%	bada	13.2	2.7%	183.1%
Windows Phone	2.5	1.6%	-14.0%	Windows Phone	6.8	1.4%	-43.3%
Others	1.8	1.1%	117.9%	Others	5.4	1.1%	14.4%

Source: Canals estimates © Canals 2012

**Figure 2.2:** Smartphone shipments for 2011 by platform, as estimated by Canals<sup>1</sup>.

The Android platform is growing very fast, with 850,000 new Android devices (including smartphones and tablets) getting activated every day. The total number of Android devices activated is greater than 300 million. More than 800 different Android devices have been released overall. The Android market contains more than 450,000 apps and has recorded more than one billion app downloads per month<sup>2</sup>. Apple's iOS devices are also very successful. Including the first quarter 2012 (October to December 2011), more than 183 million iPhones and 55 million iPads have been sold (source: Apple's quarter result reports since 2007<sup>3</sup>).

## 2.2 The Model-View-Controller (MVC) Pattern

The Model-View-Controller (MVC) Pattern is a software design pattern. The MVC pattern was first introduced by Trygve Reenskaug<sup>4</sup>, while he was working on the Smalltalk programming language at Xerox PARC. Although there are many different variations to the model view controller pattern, the core concept is the same: To separate the data model, event handling and user interface of software from one another. The variations of the pattern differ in how the components communicate with each other. The following description is based on the work by Freeman et al. (2004).

<sup>2</sup><http://googlemobile.blogspot.com/2012/02/androidmobile-world-congress-its-all.html>, retrieved Feb. 27, 2012

<sup>3</sup><http://www.apple.com/pr/>, retrieved February 3, 2012

<sup>4</sup><http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>, retrieved March 9, 2012



**Model:** The model holds all data and the state of the application. Additionally, it contains the core application logic. The model does not explicitly communicate with the view or the controller, but it has an interface which makes it possible for the view and the controller to read or modify the state of the model. Often, the model and the view are arranged in an observer pattern (the view being the observer, the model being observed), in which case the model sends notification of changes to the observing view.

**View:** The view is a visual representation of the model (or parts thereof). The data it displays is retrieved from the model's state, which the view accesses through the model's interface. The view can observe the model and update the displayed data for the user as soon as it receives a notification of a change in the model's state. As the view represents the UI, it also receives all user inputs, which are directly passed on to the controller.

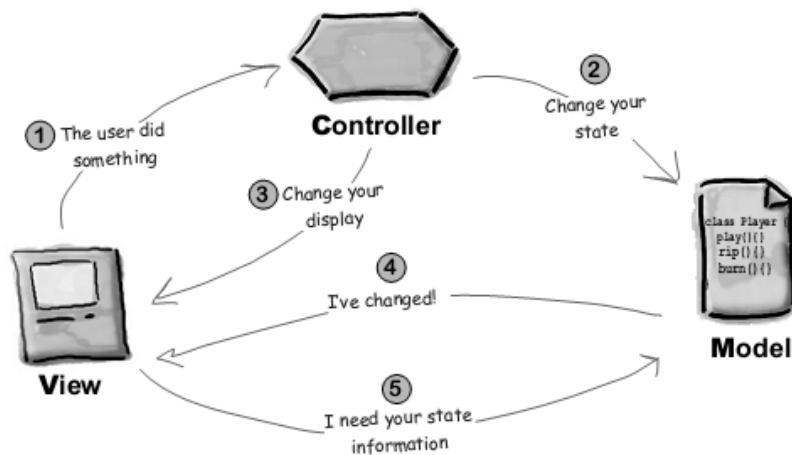
**Controller:** The controller receives user input from the view. It then evaluates the input and modifies the model's state if it should be necessary. The controller can also modify the view, if for instance certain buttons need to be enabled or disabled as a result of the user's actions.

The Model-View-Controller pattern is commonly used for the design of complex software architectures. Many frameworks and SDKs, such as Java's Swing or Apple's Cocoa use the MVC pattern as a central concept.

## 2.3 Screen size as a limiting factor

While the range of mobile devices available keeps increasing, there seems to be a limit for their screen size. The smallest smartphones available today (for the platforms investigated in this thesis) still have a display diagonal of at least 2.5 inches. One factor that limits smaller devices is touch input.

Firstly, UI elements for touchscreens need a certain minimum size in order to be easily touchable for the users without errors (see section 3.2 for size recommendations on various mobile platforms). This warrants special attention when switching from designing desktop applications to mobile applications, as touchscreens have a



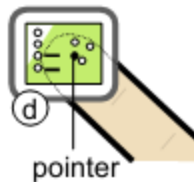
**Figure 2.3:** Control flow of the MVC pattern as described by Freeman et al. (2004).

The view receives user input and passes it on to the controller, which will initiate state updates on the model and / or view updates. Additionally, the model and view are using the observer pattern, so the model will notify the view of a state change. The view may read the model's status and update itself accordingly.

lower precision than mouse input. Sears and Shneiderman (1991) have found the error rate for small targets on touchscreens to be higher than with mouse input. Holz and Baudisch (2011) have developed a model which can reduce the error offset from touch devices down to 1.6 millimeters (down from 4) by adjusting to the model that users perceive the target on the top of their fingers while capacitive touchscreens use the contact area of the fingers, which is located at the bottom. However, the proposed models requires a camera above the target and most touchscreen devices available on the market today can't reach such a high precision yet. The findings of Holz and Baudisch (2011) suggest that with current touch-area based models, the minimum size of a touch target, to achieve an accuracy of at least 95%, is 7.9 millimeters.

Secondly, and more importantly, touching the screen - sometimes even with two or more fingers - can obstruct a large part of the user interface. This problem has been labeled the "fat finger problem" (Siek et al. (2005)). It can be mitigated by using a stylus instead of the fingers. Devices using fingers instead of a stylus seem to be a lot more popular at present, so offering the users a stylus is not always a suitable workaround to this problem.

One approach to this problem taken by Baudisch and Chu (2009) is to move the touch input to the back of the screen. This way, devices can be very small and users can interact with the UI using their fingers while the screen remains unobstructed. Because of the softness and the size of fingertips, users also need some sort of feedback to where exactly they touched the screen (or in this case the back of the device). For this reason, a pointer was added to the screen which shows the exact position of the finger on the back of the device (see figure 2.4).



**Figure 2.4:** A very small device using back-of-device interaction. The user interacts with the interface through a touch-sensitive area on the back of the device. A pointer on the screen informs the user of the exact position of his finger (pseudo-transparency). Source: Baudisch and Chu (2009)

Baudisch and Chu (2009) have shown in a study that traditional touch input fails for devices with a display diagonal smaller than one inch, while back-of-device interaction still works fine for the same size. Baudisch et al. (2008) have submitted a patent application for this technology.

A related patent application has been submitted by Zalewski and Nicholson (2009) for Sony Computer Entertainment America. This patent is not focused on the input method, but on the device in general. A touch pad on the back side of the device's case is, however, explicitly mentioned in the patent's claims. This technology is currently in use with Sony's hand-held video game console PlayStation®Vita.

## 2.4 Usability

Different definitions exist for the term usability. Bevan (1995) offers two different definitions: Usability can be seen as the ease of use of a user interface or as the ability of a product to be used for its intended purposes. Nielsen<sup>5</sup> defines usability as a quality attribute which refers to the ease of use of user interfaces and also to methods which can be used for improving said ease of use during the design process. Nielsen also identifies five quality components which define and contribute to usability:

- **Learnability:** The ease of use of a user interface for users who encounter it for the first time
- **Efficiency:** How efficiently and quickly users can accomplish tasks once they have learned how to use the UI
- **Memorability:** How long it takes users to become proficient at using the UI after longer periods of not using it.
- **Errors:** The number and severity of errors that users make and how users can recover from these errors.
- **Satisfaction:** How enjoyable users find it to use the UI.

Another attribute which can be viewed separately from usability rather than as a component of it is utility. Utility is defined as the functionality of a user interface and whether it can do what the users need. Bevan (1995) further states that usability can contribute to the quality of software and that user-centered design needs to be employed when designing user interfaces. This means that the needs and requirements of the end users should be the key factor considered when designing UIs, rather than technical factors or the underlying implementation. Both Nielsen and Bevan stress that usability must be considered at all stages of the design and development process. Only performing usability tests on a finished UI is most often not enough, as major changes will be too hard or expensive to make at this stage of the development process.

---

<sup>5</sup><http://www.useit.com/alertbox/20030825.html>, retrieved March 7, 2012

Of course all of these things hold true in the context of mobile devices as well. There are, however, different things to be considered specifically on mobile devices. These include the touchscreen input and various limitations and possibilities that depend on the platform and device, such as sensors, hardware buttons or common ways of interaction on the platform. This thesis will further investigate the special usability requirements of mobile devices and how to achieve the design and development of well-usable mobile user interfaces.

## 2.5 Usability evaluation methods

Simply putting thought into the UI design and trying to adopt a user-centered design is helpful, but by itself it may not be enough to ensure good usability of a product. Usability must be evaluated, ideally many times across the design and development process and starting even before prototyping.

The term usability evaluation refers to all methods that can be used to evaluate and improve usability. In general, all these methods aim to identify problems and flaws in user interfaces and to find possible improvements which make the software more usable for the end users. There are two categories of evaluation methods: Usability inspection methods and usability test methods. The difference is that with the former, usability experts examine the UI and try to identify usability flaws and issues while with the latter actual end users test the UI. An overview and a comparison of some of the most important usability evaluation methods can be seen in figure 2.5.

The following sections will briefly describe some usability inspection and test methods. There are more methods than those listed here, but discussing all of them would be beyond the scope of this thesis. The two methods that were used in the practical work which accompanied this thesis are discussed in more detail in section 4.5.

The information in the following sections, unless otherwise noted, is based on the research of Nielsen (1994) and Holzinger (2005).

	Inspection Methods			Test Methods		
	Heuristic Evaluation	Cognitive Walkthrough	Action Analysis	Thinking Aloud	Field Observation	Questionnaires
Applicably in Phase	all	all	design	design	final testing	all
Required Time	low	medium	high	high	medium	low
Needed Users	none	none	none	3+	20+	30+
Required Evaluators	3+	3+	1-2	1	1+	1
Required Equipment	low	low	low	high	medium	low
Required Expertise	medium	high	high	medium	high	low
Intrusive	no	no	no	yes	yes	no
<b>Comparison of Usability Evaluation Techniques</b>						

**Figure 2.5:** A comparison of various usability inspection methods and usability test methods, as outlined by Holzinger (2005)

### 2.5.1 Usability inspection methods

Usability inspection methods are performed by usability experts and aim to find problems in user interfaces and try to find ways to improve the UI. In order to do so, the interface is thoroughly inspected using established standards.

#### Heuristic evaluation

Heuristic evaluation is a usability inspection method which is relatively cheap and quick to perform. A heuristic evaluation is performed by a small group of usability experts (Nielsen recommends around five people). The evaluators closely inspect and test both the user interface as a whole and individual UI elements against a set of established usability standards, the so called heuristics.

These heuristics are usually formulated in a very general way, so that it is up to the usability experts when and how to apply them to the actual UI.

In most cases, the usability experts examine the UI independently from one another, documenting and rating them before all findings are aggregated and discussed among all experts.

## **Cognitive walkthrough**

Cognitive walkthroughs are similar to heuristic evaluations. Again, a small number of experts examine the UI. In contrast to the heuristic evaluation, the cognitive walkthrough tries to simulate the tasks of actual end users. The usability experts collect a number of tasks which end users have to perform and try to simulate the users' behavior step by step while also trying to consider limitations such as the users' knowledge and memory.

## **Pluralistic walkthrough**

End users, developers, designers and usability experts meet for a pluralistic walkthrough, where they step through given tasks and analyze and discuss the user interface and its elements.

## **Action analysis**

Action analysis is concerned primarily with the time it takes users to perform tasks. Within action analysis, as outlined by Card and Moran (1980), user interactions are broken down to single actions, such as keystrokes, mouse movements and clicks. The times needed for each action are calculated and examined.

## **2.5.2 Usability test methods**

Usability test methods have the big advantage of working with actual end users, thus producing direct and relevant feedback on the UI and problems users may have with it. The disadvantage of test methods is that they typically require more time and effort and can be more costly than inspection methods.

Three of the most common test methods are described here.

### **Thinking Aloud Test**

Thinking aloud tests (TA tests) are carried out with a small number of test users. The tests are carried out individually for each test user. A number of increasingly complex and difficult tasks are given to the test users, who are then prompted to

complete them and speak all their thoughts out loud.

Thinking aloud tests can therefore give direct insights into the users' thoughts and their view of the system, but also on the reasoning behind their actions.

While thinking aloud tests can give a lot of feedback, they do have disadvantages. TA tests take a lot of time and resources to plan, perform and evaluate. Despite trying to simulate the actual use of the system, TA tests are still unnatural to a certain degree. The users are given specific tasks, which might introduce a bias. The test users are under observation and therefore perform better than they would under normal circumstances.

### **Field Observation**

During a field observation, actual end users of a system are observed on-site at their workplace. The observers should be unobtrusive in order to enable the users to work as they normally would. Any problems the users might encounter are documented by the observers and subsequently analyzed.

Video recording and logging of user data can be used to support this method.

### **Questionnaires**

There is also the possibility to gather feedback from end users through questionnaires. Creating questionnaires which yield unbiased, meaningful data can be a difficult task and requires expertise.

In contrast to the other test methods, questionnaires are not intrusive, meaning test users don't have to be observed directly. The users' opinions and feedback can be gathered with relatively little effort. However, they should be viewed as such and can be no replacement for observations of the users' actual behavior.

Questionnaires can be combined with other methods as well. For instance, thinking aloud tests can be accompanied by a questionnaire or an interview conducted after the end of the actual test.



## 3. Related Work

### 3.1 Definitions

Some definitions of basic terms used in the following sections will be briefly explained here.

**Screen size:** The term screen size refers to the physical size of a screen, or more specifically the physical screen space that is available to display content (which sometimes slightly differs from the size advertised by the manufacturer). Following the most common practice, screen size will refer to the diagonal of the viewable screen and be given in inches.

**Resolution:** Refers to the number of distinct pixels that can be displayed in both dimensions of a screen, e.g. 1280 x 800 pixels. While the term pixel dimensions would be more accurate, resolution is the commonly used term.

**Pixel density (Screen density):** Describes the resolution (in pixels) within a specified space. The units to be used are pixels per inch (PPI) or dots per inch (DPI), which both refer to the number of distinct pixels that can be displayed on a screen across a width, height or diagonal of one inch.

**Display orientation:** The orientation of the screen as seen by the user. The display orientation is either landscape (wide) or portrait (tall). Different mobile devices can have different default orientations and can often change between orientations at runtime whenever the user rotates the display.

**Aspect ratio:** Defines the ratio between the width and the height of a display. The higher number is given first, regardless of display orientation. So a 480 x 800 pixel display with portrait orientation will still be referred to as 5:3 ratio. The aspect ration can also be given as one decimal number. 5:3 in this case would equal 1.66.

## 3.2 Mobile platforms

In the following sections some of the most common mobile platforms available will be discussed. For each platform, the general means of defining layouts will be presented. The possibilities for scaling the UI will also be examined; including automatic scaling by the platform and tools offered to the developer to support multiple screen sizes (if available).

While most platforms offer a variety of different ways to define UIs, such as OpenGLs for high performance 2d- and 3d-graphics, only the basic UIs, as used for most apps, are discussed here.

## 3.3 Android

The Google Android platform is one of the most widely spread smartphone platforms today with a spread of 48% and market leadership in 35 of the 56 countries where it is available (Leske and Cowell (2011)). While releasing Android 3.0 "Honeycomb", an Android version tailored to support tablets and other devices with larger screens in early 2011, Android has offered ways to support different screen sizes from the beginning.

Google has also released Google TV - a Smart TV platform based on Android - in 2010. Google TV will not be covered in detail, but with Android being available on TV screens as well it stresses the importance of scalable user interfaces.

### 3.3.1 Available devices

A wide variety of Android devices from many different manufacturers are available today. There is also a wide variety in different displays. The official minimum

requirements for Android devices have undergone changes over time, with updates being made with every new release of the platform. The requirements for the devices' screens have become more detailed and a bit more restrictive over time, but have more or less stayed the same. For Android 4.0 (codenamed "Ice Cream Sandwich"), the compatibility definition document<sup>1</sup> (CDD) lists these requirements with regard to the screen:

- The physical diagonal screen size of devices must be at least 2.5 inches.
- The screen size must be at least 426 x 320 density-independent pixels (see section 3.3.2), which equals the current definition of "small" screens. When combined with the minimum screen density of 120 dpi, this gives a minimum display resolution of 320 x 240 physical pixels (QVGA).
- Devices must report one of the four standard screen sizes of the Android UI framework: Small, normal, large or xlarge. The corresponding minimum display diagonals (when assuming standard densities) can be found in table 3.2.
- The aspect ratio of the screen must be between 1.3333 (4:3) and 1.85 (16:9).
- Screen densities defined by the Android UI framework are 120 dpi (ldpi), 160 dpi (mdpi), 213 dpi (tvdpi), 240 dpi (hdpi) and 320 dpi (xhdpi). The actual values of devices may vary, but they must report one of these densities to the UI framework (usually the numerically closest). The tvdpi density is meant for television screens and will therefore not be included in further discussions on screen size.
- Screens must be capable of rendering 16-bit color graphics. The capability to render 24-bit color graphics is recommended.

---

<sup>1</sup><http://source.android.com/compatibility/4.0/android-4.0-cdd.pdf>, retrieved Feb. 10, 2012

Screen size	Minimum dimension (in dp)
small	426 x 320
normal	470 x 320
large	640 x 480
xlarge	960 x 720

**Table 3.1:** Screen size standards of the Android UI framework with the corresponding minimum density-independent pixels

Android screen size	Minimum screen diagonal
<b>small</b>	3.33
<b>medium</b>	3.55
<b>large</b>	5.00
<b>xlarge</b>	7.50

**Table 3.2:** The minimum physical diagonal screen size in inches for all the Android screen size definitions.

Because of the use of the density-independent pixel unit, UI designers can basically work with four default screen sizes, as detailed in table 3.2. For instance, medium size screens must be between (approximately) 3.55 and 4.99 inches in diagonal, which includes most smartphones available on the market today. It is noteworthy that the diagonals were calculated using the default densities as defined by the Android framework. Since device manufacturers may use displays with different densities, these values are not precise (hence the difference between the 2.5 inches minimum requirement and the 3.33 inches for small displays - higher densities than the default yield smaller displays).

density \ size	small	medium	large	xlarge
<b>ldpi</b>	320 x 240	353 x 240	480 x 360	720 x 540
<b>mdpi</b>	426 x 320	470 x 320	640 x 480	960 x 720
<b>hdpi</b>	640 x 480	705 x 480	960 x 720	1440 x 1080
<b>xhdpi</b>	832 x 640	940 x 640	1280 x 960	1920 x 1440

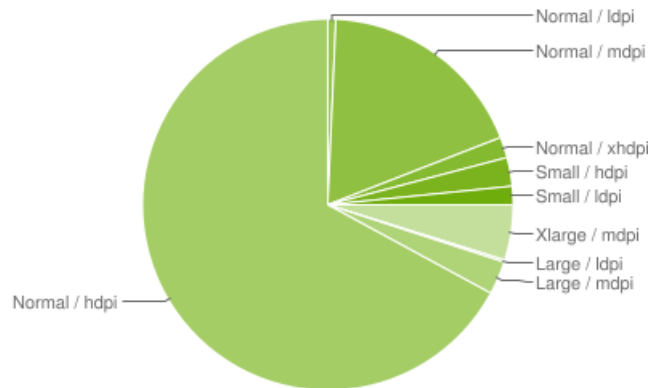
**Table 3.3:** The minimum screen resolution in physical pixels for various density / size combinations

density \ size	small	medium	large	xlarge
ldpi	1.6%	0.7%	0.2%	
mdpi		18.4%	2.9%	4.8%
hdpi	2.5%	67.1%		
xhdpi		1.8%		

**Table 3.4:** Detailed breakdown of screen size shares from figure 3.1

When looking at the display resolution, considering all possible combinations of screen sizes and screen densities gives a total of 16 configurations (see table 3.3). Since the resolutions may also be anything in between (or above) these normalized values, UI designers must expect a wide range of resolutions. Since the main factor for different UI layouts should be the screen size, not the resolution, this mostly means that graphics must scale well across all resolutions (see section 3.3.3).

Google has also released statistics<sup>2</sup> on what configurations are the most common among Android devices (see figure 3.1 for a pie chart and table 3.4 for a detailed breakdown). These figures show that 67.1% of all devices use the most common configuration (medium and hdpi) and medium currently is the dominant screen size (with a total of 88%).



**Figure 3.1:** Share of individual screen size / display density configurations of all Android devices. Data source: Google Developer Docs, <http://developer.android.com/resources/dashboard/screens.html>, retrieved Feb. 11, 2012. Data was collected from all devices that accessed the Android Market within a 7-day period ending Feb. 1, 2012

<sup>2</sup><http://developer.android.com/resources/dashboard/screens.html>, retrieved Feb. 11, 2012

Even with these things considered, Android still leaves UI designers with the problem of having to support a variety of different screen sizes. The following sections will cover some of the tools Android offers to support different screen sizes.

### 3.3.2 Supporting different screen sizes and densities

Android provides an API that allows the developer to define UIs for specific screen sizes and densities. The Android system then automatically adjusts the UI of the application to the screen it is displayed on. Should the developer decide not to provide different UIs for different screens, Android will scale and resize the UI automatically. This means that android applications will run on screens of any size regardless of whether the developer has created specific UIs for different screens. However, an automatically resized and scaled UI is not optimized for the user experiences on different devices (such as smartphones and tablets) and the Android documentation specifically recommends optimizing the UI for different screen sizes and densities<sup>3</sup>.

#### Density-independent pixel (dp)

The Android API introduced the density-independent pixel unit as a virtual pixel unit for defining layouts for different screen sizes in a density-independent way. As specified in the Android documentation<sup>3</sup>, one density-independent pixel (dp) equals one physical pixel on a screen with a density of 160 dpi, which corresponds to a "medium" density screen by Android's standards. The system uses the dp units in the layout definitions and scales them based on the density of the actual screen at runtime. Density-independent pixels are converted to actual screen pixels according to the following equation:

$$px = dp * (dpi/160) \tag{3.1}$$

So for instance, on a screen with extra high density (xhdpi, 320dpi) one dp equals two physical pixels.

Defining layouts and positions using density-indepent pixels therefore ensures that

---

<sup>3</sup>[http://developer.android.com/guide/practices/screens\\_support.html](http://developer.android.com/guide/practices/screens_support.html), retrieved January 20, 2012

UI layouts are rendered the same way on devices with the same (or very similar) screen sizes, but different screen densities.

### **3.3.3 Scaling images**

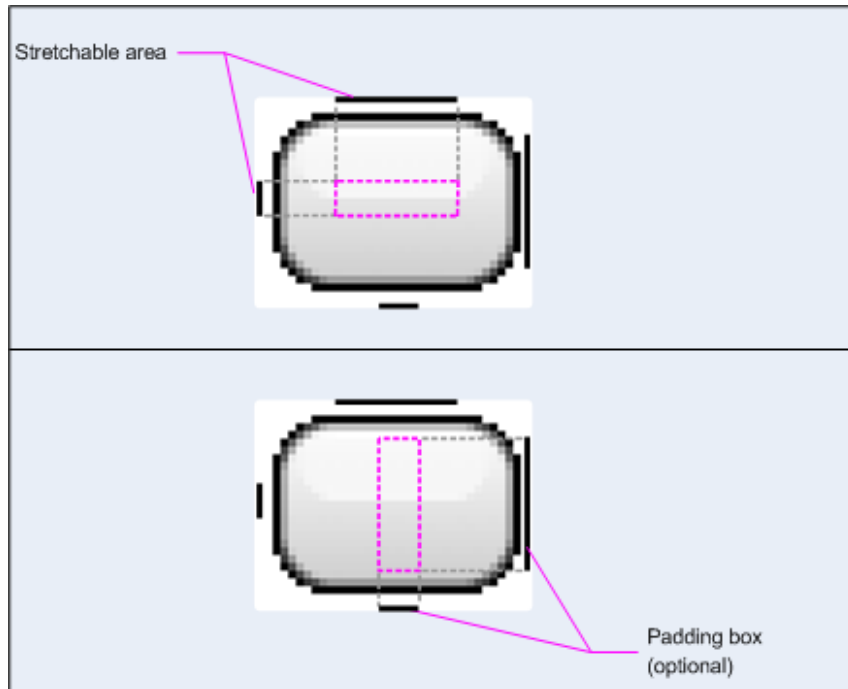
Besides layouts themselves, images are important when it comes to scaling a user interface. Applications typically use a variety of image files for their user interfaces, such as icons or button backgrounds. These images need to be handled properly when dealing with different screens; otherwise a small image might look pixelated on a large screen or a large image might take up unnecessary space because it has to be saved at full size but it gets scaled down significantly.

A very good way to deal with scaling images is to use scalable vector graphics (SVGs). However, Android currently does not support SVGs. Android does, however, offer a way to create stretchable images called Nine-Patches.

### **3.3.4 Nine-Patches**

Nine-Patches are PNG images which include additional information for Android on how to scale the image. They have the file extension ".9.png". Nine-Patches are used for images such as button backgrounds, for which the border should remain the same size while the middle should be stretched. Other image resources, such as icons are not suited for Nine-Patch definitions. A Nine-Patch Image is divided into nine (three times three) sections, the outer ones representing the border of the image and the middle one being the stretchable content. Android will stretch only the middle part, leaving the borders untouched.

Android offers a graphical editing tool to define Nine-Patch images. The tool uses lines on the top and left of the image to specify the stretchable area of the image and optionally also lines on the bottom and the right of the image which specify a padding box, which is the area where text should be placed in the image.



**Figure 3.2:** A sample Nine-Patch definition. The lines on the top and left define the stretchable area. Only this area will be stretched to fit on larger screens, while the border will remain the same (having a one pixel thick border). The lines on the bottom and the right define the padding box. If the image (and its correspondig view in Android) contains text, the text will be fitted into the padding box.



**Figure 3.3:** A button with a Nine-Patch background, scaled to two different sizes.

### 3.3.5 The Back Stack

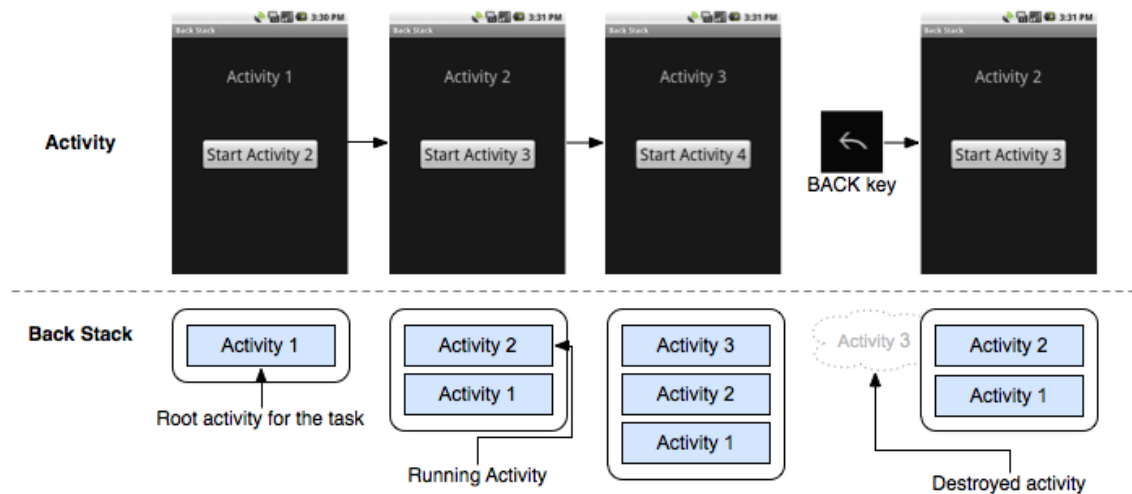
Android applications usually consist of multiple activities. An activity provides a screen or windows and allows the user to perform a certain task, such as writing an email, viewing images or dialing the phone.

Naturally, when an application is used, activities are switched many times. When a new activity is started the previous activity is stopped and its state is being saved to the "back stack" or "activity stack". The new activity is also pushed onto the back stack and takes the focus. When the new activity is completed or when the user



presses the back key on the phone or tablet, the current activity is popped from the stack and destroyed and the previous activity resumes where it left off<sup>45</sup>.

Whenever the user wants to go back to where he was before, he can simply push the back button. In addition to the functionality itself, this feature is also relevant from a UI design point of view. A designer can choose to split a large application (e.g. an existing desktop application) into multiple smaller activities and allow easy navigation between them using the back stack.



**Figure 3.4:** An illustration of Android’s back stack. Whenever a new activity is started, it is pushed to the stack, taking the focus. When the user presses the back button or the activity finishes or closes, the current activity is popped from the stack and destroyed while the previous activity is restored.

### 3.3.6 Fragments

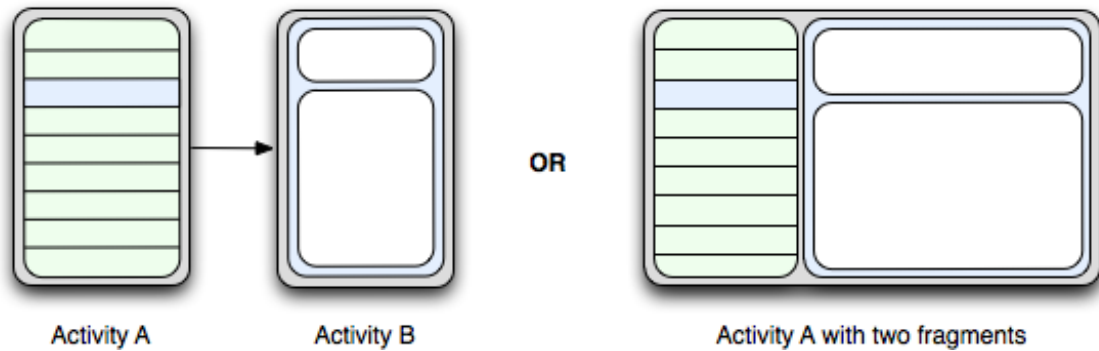
In order to support scalability of applications, especially between smartphone and tablet screens, Android introduced the concept of fragments. A fragment represents a part or a behavior of the user interface. Multiple fragments can be included in one activity to build a large, multi-pane UI. An activity can also consist of a single fragment only and fragments can be used in multiple activities.

Typically, a smartphone screen will display most fragments individually, with each one taking up the entire screen. On a larger tablet screen, these fragments might

<sup>4</sup><http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html>, retrieved January 12, 2012

<sup>5</sup><http://developer.android.com/guide/topics/fundamentals/activities.html>, retrieved January 12, 2012

be combined and displayed at the same time, in one activity. For instance, a news application can show a fragment with a list of current news on the left side and the currently selected news article in another fragment on the right side. On a smartphone, the news list and the news article would be displayed in separate activities<sup>6</sup>.



**Figure 3.5:** Android fragments in action. The figure on the left hand side shows a smartphone screen where two fragments are displayed on the whole screen, one after another. The figure on the right shows a larger tablet screen, which combines the two fragments.

### 3.3.7 Android Design

In the past, various documents have been created on the subjects of Android UI design and user experience (e.g. <sup>7</sup>, but they were - despite being created by Google employees - never made officially available on the Android Developer website.

In January 2012, following the release of Android 4.0 ("Ice Cream Sandwich"), an official website containing design guidelines, named "Android Design"<sup>8</sup> has been launched.

The guidelines contain some recommended patterns, such as the action bar pattern (figure 3.6), which are intended to establish a common look-and-feel for Android applications. Some of the screen size related features mentioned above are also part of the guidelines.

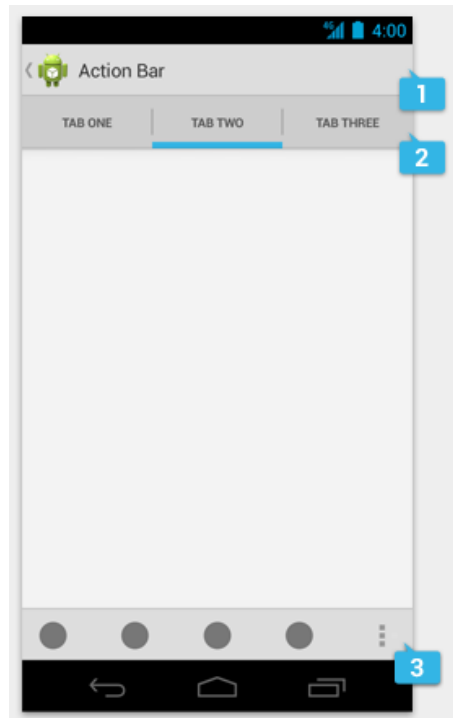
With regard to the size of components the guidelines recommend 48 density-independent pixels as a minimum. 48 dp equal roughly 9 millimeters, which should be enough

<sup>6</sup><http://developer.android.com/guide/topics/fundamentals/fragments.html>, retrieved January 12, 2012

<sup>7</sup><http://tinyurl.com/2umdmkg>, retrieved Feb. 12, 2012

<sup>8</sup><http://developer.android.com/design/>, retrieved Feb. 12, 2012

for components to be touched easily and accurately. It is further recommended to design the whole layout in "steps" of 48 dp with spacings of at least 8 dp between controls.



**Figure 3.6:** A simple example which illustrates the action bar pattern. The action bar (1) is on top and should contain the app logo and important actions and navigation consistently across the app. The top bar (2) can contain navigation between views in one activity and the bottom bar (3) can contain actions related to the current view. Source: Android Design Guidelines<sup>9</sup>

---

<sup>9</sup><http://developer.android.com/design/patterns/actionbar.html>, retrieved Feb. 12, 2012



**Figure 3.7:** The Windows Phone 7 home screen with some live tiles which update in real time. Image source: Windows Phone 7 website<sup>10</sup>

## 3.4 Windows Phone

The Windows Mobile operating system has initially been released in 2000 for PDAs and Smartphones. Many different devices were available, most of which offered touchscreen input through a stylus.

In late 2010, the successor of Windows Mobile, Windows Phone 7, was launched. While Windows Mobile has previously been successful in the business sector, Windows Phone 7 was explicitly aimed at the consumer market in an attempt to have a share of the great success of Android and iOS. All Windows Phone 7 devices offer capacitive, multi-touch capable touchscreens. A new user interface, called Metro, was introduced. One feature of Metro are the rectangular tiles which can be placed on the home screen (see figure 3.7) and give the UI a distinctive look.

Several third-party manufacturers offer smartphones with Windows Phone 7, including HTC and Nokia.

### 3.4.1 Screen sizes

The range of screen sizes available for current Windows Phone 7 devices is quite small as Microsoft has specified that all devices must have a WVGA format display

---

<sup>10</sup><http://www.microsoft.com/windowsphone/en-us/default.aspx>, retrieved Feb. 12, 2012

with a resolution of 800 x 480 pixels (Microsoft). Screens with a resolution of 480 x 320 pixels (HVGA) are also expected for the future (Petzold (2010)).

However, the screen size can still be an issue when porting existing Windows (desktop) applications to Windows Phone. In the future this might be more of an issue with Microsoft having announced an App Store for Windows 8 and an App Store already available for XBox360 videogame consoles. Applications running on all these platforms will have to support screens from smartphone size over PC monitors up to TV screens.

The two different resolutions on Windows Phone 7 devices also have different aspect ratios: WVGA has an aspect ratio of 5:3 (1.66) while HVGA has an aspect ratio of 3:2 (1.5). While this is not a huge difference it should still be considered when designing user interfaces for both resolutions.

### **3.4.2 Defining layouts**

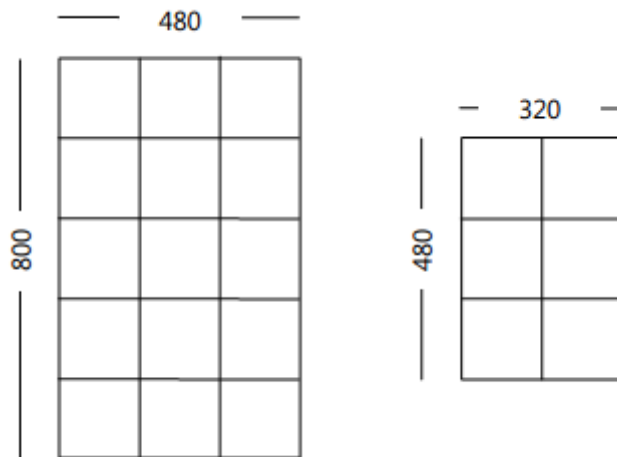
The Windows Phone platform uses a specific version of the Silverlight application framework in which user interfaces are declared using Extensible Application Markup Language (XAML) files.

Sizes of UI elements, layouts and fonts are simply measured in pixels. Additionally, elements can also be sized using the "Auto" property, which makes them take up all available space horizontally and/or vertically. Windows Phone does not have any other means of defining those values (such as percent), so the designer must tailor the UI to the device's resolution.

### **3.4.3 Supporting different screen sizes**

The Windows Phone platform offers no special ways or tools to deal with the different screen sizes. The Programming Guide (Petzold (2010)) suggests that programs adapt themselves to the screen size using conditional code branches and different XAML layout files for size-dependent layouts.

Furthermore, it recommends dividing the screens into squares of 160 x 160 pixels, which is the largest common denominator for the two screen sizes. The HVGA screen has 2 x 3 of these squares while the WVGA screen has 3 x 5.



**Figure 3.8:** The two resolutions of Windows Phone 7 along with the suggestion to divide the screen into 160 pixel squares for visualization and UI design. Source: Petzold (2010)

## 3.5 iOS

iOS is a mobile operating system created by Apple. There are currently three series of devices which use iOS: The iPhone, the iPod Touch and the iPad. Some key features of iOS with regard to user interfaces include multi-touch capable touchscreen support, touch gestures (such as pinch zoom) and high responsiveness.

The iPhone is a smartphone which was first introduced in 2007 and contributed greatly to the development and propagation of smartphones. Apple has since released a total of four improved versions of the iPhone and had great commercial success with the device series, selling more than 146 million iPhones until the end of 2011. In 2010, Apple presented the iPad, a tablet computer which also runs iOS, therefore having very similar user interfaces as the iPhone. Also running iOS and released in 2007 was the iPod Touch, the newest series of Apple's mobile digital multimedia player.

The second generation of Apple TV, a set-top box for the playback of digital multimedia content on television screens, also uses a modified version of iOS, thus expanding the use of iOS beyond mobile devices. The focus in the following section will be on the mobile iOS devices, i.e. the iPhone, iPad and iPod Touch.

Device	Screen size	Resolution	Aspect ratio
iPhone (up to 3G)	89mm	480 x 320 pixels	3:2
iPhone (4 and 4S)	89mm	960 x 640 pixels	3:2
iPod Touch (up to 3rd gen.)	89mm	480 x 320 pixels	3:2
iPod Touch (4th generation)	89mm	960 x 640 pixels	3:2
iPad	250mm	1024 x 768 pixels	4:3

**Table 3.5:** Comparison of the screen sizes of iOS devices

### 3.5.1 Screen sizes

Since there are only three device series for iOS which are all designed and manufactured by Apple, the differences in screen sizes, resolutions and aspect ratios are manageable. As can be seen in table 3.5, the iPhone and iPod Touch screens are identical with 89mm screens. The earlier generations had 480 x 320 screens while the resolution and the pixel density have been doubled for the current versions. The iPad's screen measures 250mm diagonally and has a resolution of 1024 x 768 pixels, giving it an aspect ratio of 4:3 in contrast to the 3:2 aspect ratio of iPhone and iPod Touch. This means that UI designers have two different screen sizes to consider when developing for all iOS devices: The iPhone / iPod Touch screens and the iPad screen.

Figure 3.9 shows the iOS devices side by side, illustrating the difference in size between the iPhone / iPod Touch and the iPad



**Figure 3.9:** Size comparison of the current iPod Touch, iPhone and iPad models. Source of individual images: Technical specifications as found on [www.apple.com](http://www.apple.com), retrieved Jan. 30, 2012

### 3.5.2 Defining layouts

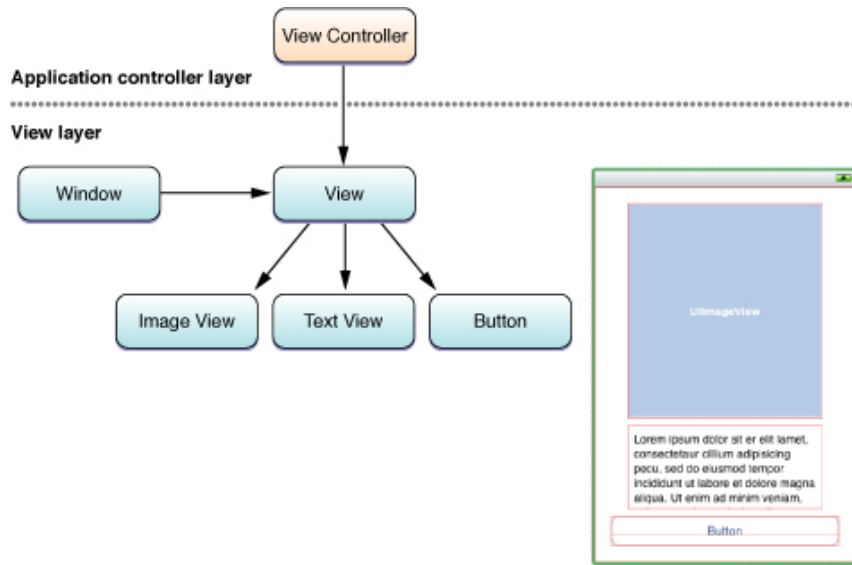
The iOS SDK, which only runs on MacOS computers, is required for creating native iOS apps. The Xcode IDE is the default development environment used for creating iOS apps.

User interfaces can be seen as hierarchies of views, as can be seen in figure 3.10. Views can be built using the built-in editor of Xcode or programmatically in the sourcecode. If Xcode is used, a so called Nib file is created. The Nib file contains the view hierarchy and layout.

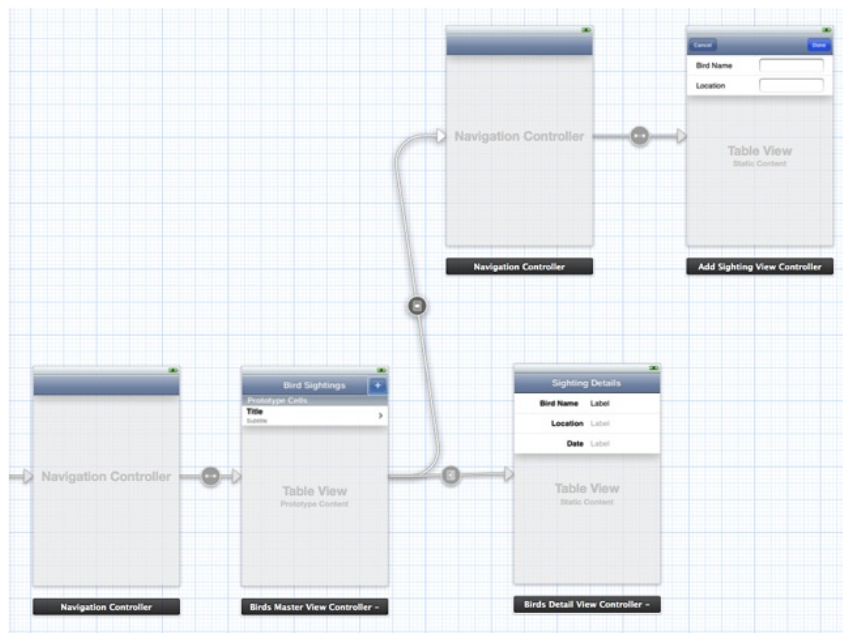
Xcode allows the visual creation of individual views or storyboards. Storyboards contain a number of view controllers with associated views which can be linked by relationships or events. Using the storyboard view, designers can create a flow of control based on the user's actions. An example can be seen in figure 3.11.

<sup>10</sup><http://developer.apple.com/library/ios/>, retrieved Jan. 30, 2012





**Figure 3.10:** The view hierarchy of a very simple iOS application. The window contains a single view, which in turn contains an Image View, a Text View and a Button. Source: Apple iOS developer library<sup>10</sup>



**Figure 3.11:** An example of a storyboard created in Xcode. The view on the very left shows the application’s UI after initialization. The arrows correspond to user actions. Therefore, the storyboard depicts a workflow within the app. Source: Apple iOS developer library<sup>10</sup>

Device	Screen size (in points)
iPhone and iPod Touch	480 x 320
iPad	1024 x 768

**Table 3.6:** Screen sizes (in points) of iOS devices in landscape orientation.  
Data source: Apple View Programming Guide for iOS<sup>11</sup>

### 3.5.3 Supporting different screen sizes

Apple offers an abstraction of pixels for coordinates, measurements and distances in UIs which is called points. A point is a floating-point value which can have different actual sizes on different devices. One point on the iPhone 4(S) screen equals two pixels while on earlier iPhone versions it equals one pixel, therefore allowing the designers to use the same point values for both devices. The minimum measurable size of points varies accordingly. The iPhone 4(S) can handle half points while older devices can only handle integer values. Future devices may have different resolutions, but the point units ensure that UIs are scaled properly.

In order to support the higher resolutions of newer devices, iOS (version 4 and above) uses different image resources (one low-resolution and one high-resolution image). Developers should provide both images so that the appropriate one can be chosen and displayed at runtime.

### 3.5.4 Apple's Human Interface Guidelines

Apple has composed a set of UI design guidelines and principles called the iOS Human Interface Guidelines. It can be found on Apple's website<sup>12</sup>. These guidelines serve two purposes: Firstly, they teach designers how to create UIs with good usability and how to avoid usability problems. Secondly, they establish a common look and feel for the platform, giving iOS applications a distinctive look when compared to other platforms while at the same time making them look and feel familiar for users.

<sup>11</sup>[https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewPG\\_iPhoneOS/ViewPG\\_iPhoneOS.pdf](https://developer.apple.com/library/ios/documentation/WindowsViews/Conceptual/ViewPG_iPhoneOS/ViewPG_iPhoneOS.pdf), retrieved Jan. 30, 2012

<sup>12</sup><http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG>, retrieved Jan. 30, 2012

Among many other things, the guidelines recommend a minimum size of 44 x 44 points for all tappable UI elements. This equals roughly 7 x 7 millimeters, which is very similar to Android's guideline.

The guidelines contain an extensive amount of information and will not be discussed further.

## 3.6 HTML and CSS

HTML5 is the latest version of HTML and is currently still under development. It is being developed by the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG). The latest version of the standard can be found on the website of the WHATWG<sup>13</sup>. All the information in this section is based on this version of the standard.

It is noteworthy that while the term HTML5 officially only refers to the latest version of the Hypertext Markup Language itself, it is commonly used to describe a whole variety of current technologies and standards, including CSS (Cascading Stylesheets), DOM (Document Object Model) and XML (Extensible Markup Language).

HTML is used to define the structure and the contents of a web page while CSS is used to define the appearance and layout of HTML documents. Thus, when designing user interfaces one must consider both HTML and CSS.

The HTML5 standard defines a number of new tags which weren't contained in its predecessors. These tags include some new tags for multimedia content, such as the `<audio>` and `<video>` tags or the `<canvas>` tag which creates a canvas that can be drawn on on the fly. Other tags are strongly related to layout, such as the `<header>` and `<footer>` tags, which define an introduction or navigation section and a footer section respectively. With these tags, HTML5 has become more expressive than its predecessors, which used the `<div>` tag for sections of a web page.

---

<sup>13</sup><http://www.whatwg.org/html>, retrieved Feb. 17, 2012

### 3.6.1 Defining layouts

HTML defines a web page's structure and content using a tree of elements and text content. Each element has a start tag and an end tag. Tags can be nested, but they may not overlap (i.e. tags nested within other tags must be closed before the outer tags). HTML is, however, intended to define only the structure of a document and not its layout.

The style (i.e. font sizes, colors etc.) and layout of HTML pages should be defined using Cascading Stylesheets (CSS). CSS can assign attributes either to content placed within specific tags or to elements with a specific ID. Among others, CSS can define the position (relative or absolute), size and the alignment of elements.

CSS offers various units for measurement. Besides pixels and length units such as millimeters, measurements can also be given in percent and in em. Em defines the relative font size of an element. For instance, an element with a font size of 0.5em has half of the normal font size. This is important for both accessibility and scalability, as it also changes font size relative to the user settings. Relative measurements can also help to make websites and web applications easier to scale.

### 3.6.2 Touch input

HTML5 also supports multi-touch input<sup>14</sup>. The touch events include events representing the start of a touch, movement of touch ("swiping") and the end of a touch. A list of all current touches along with their page or screen coordinates can easily be obtained, both for the entire document and for a given DOM element.

HTML5's touch events rely on both the mobile platforms and the browsers' support. They are widely implemented, but there are still a few browsers that don't fully support them.

There are a few APIs that abstract touch event handling, see section 3.7.2.

---

<sup>14</sup><http://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html>, retrieved Feb. 14, 2012

## 3.7 Multiplatform development

When developing for mobile devices, developers face choices and challenges. They must choose which platform(s) to develop for and they must choose their distribution channels. While every platform has its default App store (e.g. the Android Market for Android), a variety of unofficial stores exists. The Wireless Industry Partnership (WIP) AppStore Catalog<sup>15</sup> currently lists more than 120 different app stores.

Developing apps for multiple platforms has the major advantage of reaching a much larger audience. However, when developing native apps for multiple platforms, the workload and consequently the development costs are also much higher. As covered in the previous sections, every mobile platform has different base technologies for native app and UI development, including different programming languages. An overview of these programming languages can be found in table 3.7.

There is, however, also the possibility to create web apps or hybrid apps which require less effort to develop for multiple platforms. Furthermore, there are multiple frameworks for multi-platform development. Some of them enable the developers to deploy one app to several platform as a native app while some help to make an HTML web interface feel more like a native app. These two approaches - developing a web app for multiple platforms and using a multi-platform framework - are subject of the following sections.

---

<sup>15</sup>[www.wipconnector.com/appstores](http://www.wipconnector.com/appstores), retrieved Feb. 14, 2012

Platform	Programming language(s)
Android	Java, C and C++ (through Native Development Kit)
iOS	Objective-C
Windows Phone	C#
Blackberry	Java
Symbian	C++

**Table 3.7:** Programming languages used for app development on various mobile platforms

Native app advantages	Web app advantages
<ul style="list-style-type: none"> <li>• Broad access to device hardware and platform features</li> <li>• Close integration with platform and other apps</li> <li>• Better performance</li> <li>• Exponation through app store(s)</li> <li>• On-device storage</li> </ul>	<ul style="list-style-type: none"> <li>• Large target audience (mobile platforms + desktop)</li> <li>• Easy multi-platform development</li> <li>• Easy to update across all platforms</li> <li>• No certification required</li> </ul>

**Table 3.8:** Advantages of native and HTML5 web apps. Sources: Meier and Mahemoff (2011) and original research

### 3.7.1 HTML5 versus native apps

The question whether to develop a native app or a web-based HTML5 app is one which every app developer will inevitably ask himself. The different platforms and environments of mobile devices (see table 3.7) make porting an app from one platform to another a difficult task.

HTML5 is a promising alternative - every one of these platforms has a browser and the capability to run web apps.

While HTML5 apps can run on any mobile platform with little to no extra effort, native apps also have advantages (see table 3.8). Access to the system's hardware or some hardware-related features such as the camera is often only possible in native apps. Native apps can also integrate themselves seamlessly into the system: They use the native look-and-feel, can offer widgets to be displayed on the home screen, interaction with other apps and the use of system notifications. Native apps also tend to have better performance than web apps, but current browsers are getting more and more performance optimizations while at the same time newer devices offer more computational power, making the difference diminish.

It is also possible to develop hybrid applications. These are native applications which use views to display HTML5 content, which can be the same across all platforms.

### 3.7.2 Multiplatform frameworks

Since the development of apps for mobile platforms is getting more important with numerous platforms being available on the market, platforms have been developed which should facilitate easy app development on multiple platforms. These multiplatform frameworks come in many variations. Some of them only change the visual appearance of web apps while others offer whole software development kits on their own, which can build the developed apps to multiple target platforms as native apps.

These frameworks typically use established standard technology such as HTML5 and JavaScript. This can be a big advantage for developers familiar with web development as they don't need to learn to use the languages and development tools for native apps on one or more platforms.

One disadvantage of web UIs is that their controls usually look different from those of native apps. While this does not harm functionality at all, it may cause the app and UI elements to feel less familiar to the user. There are some frameworks and libraries which simply apply a native "skin" to HTML5 apps, which should ideally make them indistinguishable from native apps.

In the following sections, three multi-platform frameworks are examined. There are, of course, many more frameworks available. Criteria for the selection of the frameworks were that these were, at the time, some of the most feature-rich and most popular available. The aim was also to have frameworks with different key features in order to be able to compare and contrast them. Furthermore, the frameworks are also available for free. There are several proprietary frameworks, but they could not be tried out thoroughly and were therefore excluded from the selection.

## jQuery Mobile

jQuery Mobile<sup>16</sup> is a web app framework which was developed by the developers of the popular JavaScript library jQuery.

The main focus of jQuery Mobile lies on the design and presentation of mobile apps. The technologies used by jQuery mobile are HTML5, CSS3 and JavaScript, which enables it to run on most current mobile devices. jQuery Mobile can be used to both make web apps feel more like native apps and to apply a brand layout to them, giving them a more polished look than a default web app. The layout is scaled to the target device. Small adaptations can be made automatically. For instance, labels can be placed beside input fields on larger devices while being placed above them on smaller devices.

The jQuery mobile website offers a drag-and-drop tool for building simple UI layouts called Codiqa. Components such as buttons, text, images and lists can be dragged into the UI, allowing for rapid prototyping. The resulting source code can then be downloaded and further modified. This makes it much easier for new or inexperienced developers to get started, as the other platforms such as Android have a much steeper learning curve.

The jQuery Mobile framework is free and open source. It can be used under the terms of either the MIT License<sup>17</sup> or the GNU General Public License<sup>18</sup>. The basic version of the Codiqa, which allows the download of the source code created, is free. Advanced versions which offer additional features cost 10\$ per month for single users and 30\$ per month for teams.

jQuery Mobile offers some more features which are not strictly related to design, most notably the handling of events such as various forms of touch input (e.g. tap, swipe) and orientation change (i.e. the user turning the device from portrait to landscape mode or vice versa).

---

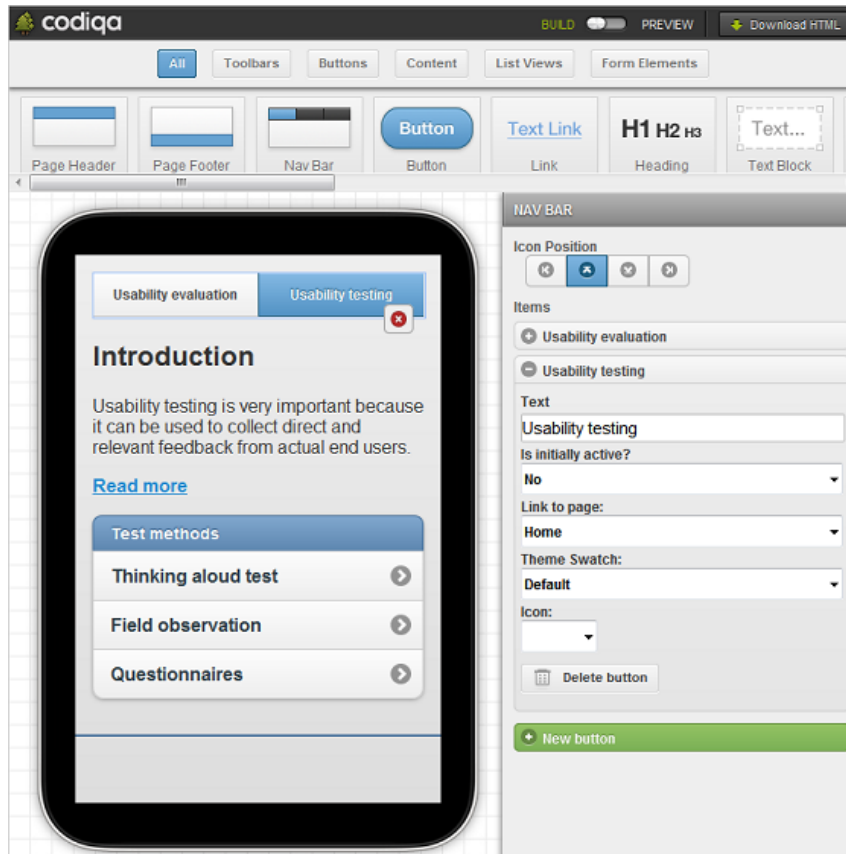
<sup>16</sup><http://jquerymobile.com/>

<sup>17</sup><https://github.com/jquery/jquery/blob/master/MIT-LICENSE.txt>, March 10, 2012

<sup>18</sup><https://github.com/jquery/jquery/blob/master/GPL-LICENSE.txt>, March 10, 2012

<sup>19</sup><http://jquerymobile.com/>





**Figure 3.12:** The Codiqa rapid prototyping tool for jQuery Mobile<sup>19</sup> enables designers to create simple user interfaces for web apps very quickly.

The platforms fully supported by jQuery Mobile include iOS (3.2 and above), Android (from version 2.1), Windows Phone 7 and Blackberry (6.0 and above). Some older versions of the platforms are partially supported. JQuery Mobile also supports a number of browsers for both mobile and desktop devices, such as Firefox, Chrome or Opera.

jQuery Mobile also offers a web application called ThemeRoller, which allows designers to easily apply themes or brandings to HTML5 websites. ThemeRoller is basically an interactive tool to create CSS3 stylesheets. It uses CSS3 properties to modify colors and fonts. Designers can create multiple themes and color schemes which can then be downloaded as CSS files.

The jQuery mobile framework offers a lot of utility to designers. Its rapid prototyping tool can serve as a good entry point for developing mobile apps, providing

a skeleton UI and the possibility to customize the look-and-feel. In most cases, however, the UI will still have to be refined. As for writing JavaScript code, it offers some methods and utility functions.

With regards to the scalability of the apps produced jQuery Mobile does offer some help by making it easy to create and modify layouts, but in the end it is up to the designer to modify and test them in order to ensure they look good on screens of different sizes.

## PhoneGap

PhoneGap<sup>20</sup> is an app platform which can build apps from a single code base for multiple target platforms as native apps. It is based on HTML5, CSS3 and JavaScript and abstracts the hardware of mobile platforms into a JavaScript API.

The library provides interfaces for hardware access and certain native features. Hardware access includes the device's accelerometer, the camera and the compass. Some of the platform features are access to the contact list or to the file system, notifications and geolocation. All of these features are available for Android, iPhone (3GS and above) and Windows Phone 7. Other platforms such as Blackberry or Symbian have a subset of these features available. PhoneGap thus manages to combine many of the advantages of native apps and web apps.

Technically, the resulting apps generated by PhoneGap are not truly native apps but hybrid apps. The apps use web views which display the HTML5 content, but they can be built to the platform's native format and therefore also distributed over several app stores.

Developers need to set up the SDKs for all platforms they want to develop for. This implies some limitations: In order to deploy build PhoneGap apps for iOS the iOS SDK is required, which in turn requires a Mac OS computer. PhoneGap supports the default IDEs for various platform (e.g. Eclipse with the Android Development Tools for Android) and only requires the libraries to be imported.

PhoneGap is an open source project and is available for free. It uses the Apache License (version 2.0)<sup>21</sup>.

---

<sup>20</sup><http://phonegap.com>

<sup>21</sup><http://www.apache.org/licenses/LICENSE-2.0.html>, retrieved March 8, 2012

## Titanium Mobile SDK

The Titanium Mobile SDK<sup>22</sup> by Appcelerator Inc. also offers the possibility to create native apps from a single codebase. It comes with a complete IDE called Titanium Studio, which is based on the Aptana Studio web development UI<sup>23</sup>. The app logic is written using JavaScript.

Much like PhoneGap, Titanium offers an extensive API with access to native hardware and software features. It allows the use of additional JavaScript libraries such as jQuery. It currently supports iOS, Android and BlackBerry and also allows the creation of mobile web apps (the latter still being in beta status).

Appcelerator also offers a marketplace<sup>24</sup> for templates, designs and modules for applications.

In contrast to PhoneGap, Titanium Mobile UIs are using real native UI components rather than web views.

Like PhoneGap, Titanium Mobile requires the SDKs of the target platforms to be installed and set up within the SDK.

It uses the Apache License (version 2.0)<sup>5</sup>, but the software is proprietary. The basic version of Titanium Mobile is free. There are also a Professional Edition and an Enterprise Edition available. These offer additional support, some additional modules and early access to new features.

The feature of creating native UIs for different platforms is very promising. The SDK was only briefly examined for this thesis and further investigation is necessary in order to evaluate how well Titanium handles the rendering of complex native UIs. While the building of native apps can be an advantage, it also means more overhead as a number of different apps needs to be built and tested with every new update.

---

<sup>22</sup><http://www.appcelerator.com/products/titanium-mobile-application-development/>

<sup>23</sup><http://aptana.com/>

<sup>24</sup><https://marketplace.appcelerator.com/>



## 4. Materials and Methods

### 4.1 Introduction

The practical work accompanying this thesis was done in cooperation with Boom Software AG (hereafter simply referred to as Boom Software). It consisted of usability tests for existing desktop and web-based software, usability consulting for a tablet application in development and the creation of usability guidelines for the developers.

The work was conducted by Michael Geier and Peter Treitler (hereafter referred to as the usability team) and supervised by Roman Bobik, project manager at Boom Software.

### 4.2 About Boom Software AG

Boom Software AG is a software developing company located in Leibnitz, Austria. It was founded in 1995 and has 50 employees (as of January 2012). The company's main products are two software solutions for businesses: The Boom Maintenance Manager and the Boom Production Manager. Boom Software advertises total customization for its customers by individually designing and customizing the software to meet specific needs and requirements. Boom Software's products are used by a total of more than 20,000 end users. ((Boom Software AG) (2012)).

### 4.3 The Boom BORA framework

The underlying technology that enables Boom Software to customize their software is the Business Oriented Rapid Adaption (BORA) Framework. The information on the BORA framework provided here is based on the BORA marketing documents, as provided by Boom Software in January 2012.

BORA was initially intended to be an internal method or design- and programming-tool. The goal was the unification of all technologies into a Microsoft .NET Framework based platform. The benefits were that employees would only need to learn to use one technology, which is the same across projects and departments, therefore allowing them to switch projects more flexibly. It was also intended to establish a base technology for projects, reducing the workload on technical problems and allowing employees to focus on the creation of content and the solving of content-related problems.

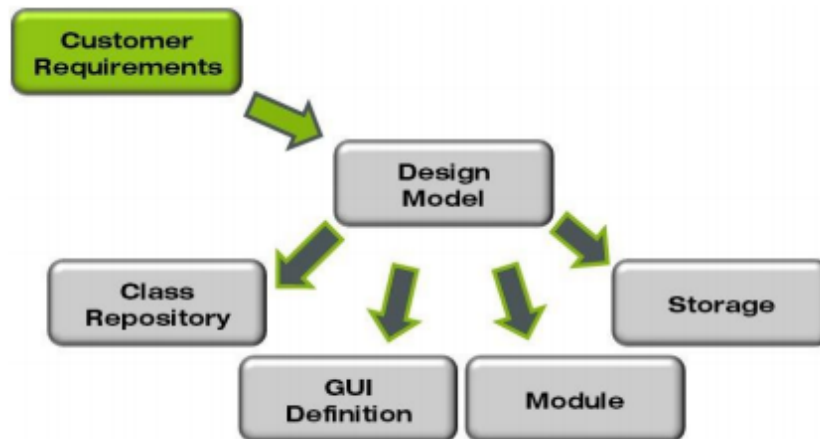
BORA firstly serves as a technological basis, implementing technical issues and offering abstract standard solutions. Furthermore, it is an application framework, providing solutions for common issues of software development, such as distributed environments, modifiable UIs etc. Finally, BORA is also a standard, both using existing standards (such as the XAML format) and offering a clearly defined procedural model, allowing the implementation of UML business models.

All of Boom Software's products tailored to customers' needs are based on BORA, including new applications as well as updates, modifications or addons to existing applications.

The BORA framework uses a modular concept, allowing the creation of reusable modules. Business logic and user interface are clearly separated from one another. The development process with the BORA framework follows a model driven architecture (MDA) concept. The UI definitions, along with other modules of the programs is generated from the design models, which in turn is based directly on the customer requirements (see figure 4.1).

The BORA framework also allows rapid prototyping, by generating a user interface

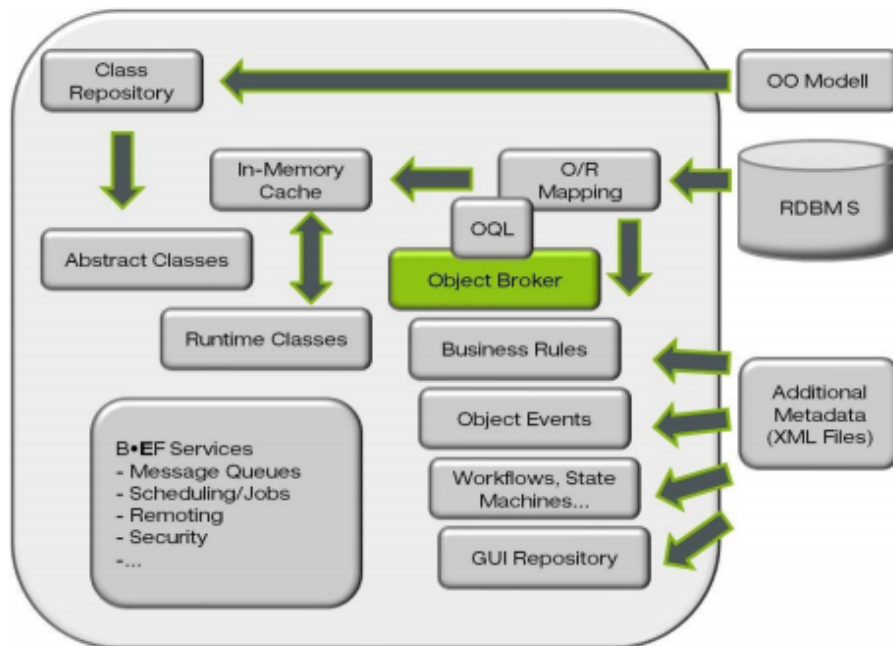
with basic functionality directly from the design model with very little effort for the developers. The prototype can then be customized and improved upon. User interfaces can be generated for Windows and for websites. An extension to mobile devices (see section 4.4) is currently in development.



**Figure 4.1:** The (simplified) BORA process model. Source: BORA marketing documents, as provided by Boom Software in January 2012

Figure 4.2 shows the core components of the BORA framework in more detail. The object oriented model is designed based on the customer’s specifications. The database and a basic UI are automatically generated. The object oriented model is translated to abstract classes, which may not be modified by the developer. The corresponding specific classes are loaded at runtime, which allows for an easier modifications of already shipped and running applications.

The usability team identified a number of usability issues. If these issues were global (i.e. affecting all of Boom Software’s applications), the fixes would be made directly in the BORA Framework, which creates a big leverage by applying the fix to all future BORA applications and possibly updates of existing applications. In cases where the issues only applied to one application, methods to avoid them were included in the usability guidelines so that developers and designers will be aware of them and improve future applications.



**Figure 4.2:** The core components of the BORA framework. The requirements (on the right) are separated from the implementation. Source: BORA marketing documents, as provided by Boom Software in January 2012

## 4.4 The Boom Mobile app

In the past, Boom Software has been producing software for Microsoft Windows systems. In summer 2011, Boom Software was approached by a customer which asked for a tablet PC version of an application which would allow employees to easily track and update maintenance tasks while performing maintenance on location. The target devices were 7 inch Samsung Galaxy Tabs running Android 2.2.

Since Boom Software had no prior experience with Android development, the task was outsourced to an external company. The team was present at meetings to provide consulting with regard to usability and to discuss design decisions.

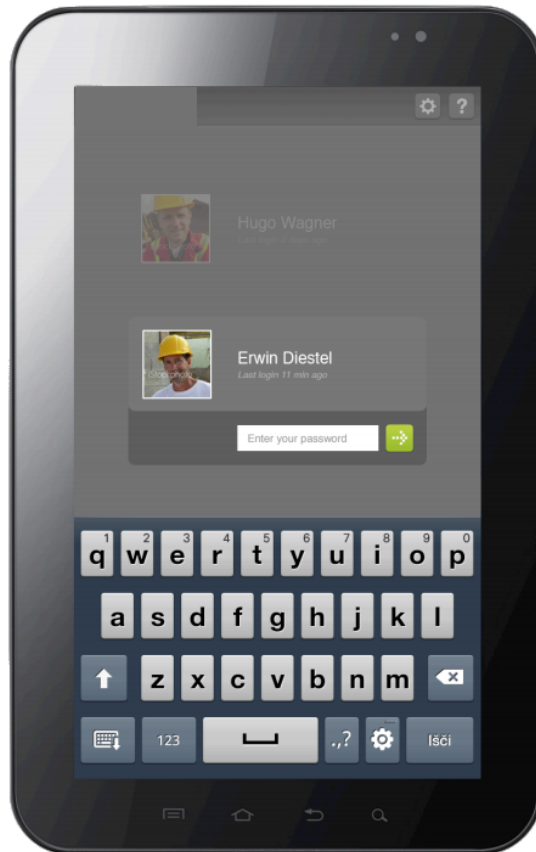
The app follows the Boom BORA framework and doesn't define its own user interface layout on the client. It receives XML data from the server, which is then rendered on the client. Therefore, changes to the user interface can be made on the server side without the need of an update on the client side.

While the app requires an internet connection in order to synchronize tasks with



the server, changes can be made offline and are automatically synchronized as soon as a connection is available again, which enables users to work in remote areas with no mobile reception.

#### 4.4.1 First prototype



**Figure 4.3:** The login screen of the Boom Mobile app.

The app consists of three main screens. The first (figure 4.3) is a login screen, which looks much like the login screen of Windows 7. A vertically arranged list of users is presented, with a user image, the user name and the time of the last login. Upon tapping an entry in the user list, the user is prompted for the password. After successful authentication, the task list is displayed (figure 4.4).

The task list, per default, shows the incomplete tasks the employee has been assigned. For each task, the due date, a brief description and its status (done or not done) are displayed.

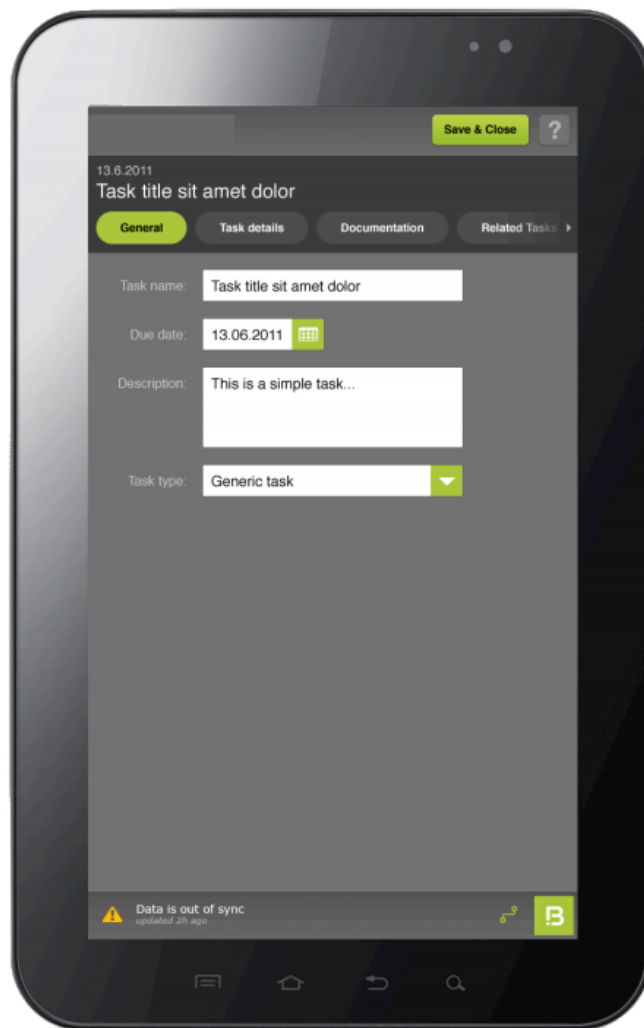
Usually, the user will only see the tasks he or she has not finished yet. When he

marks a task as finished, it will disappear from the list. On the top of the screen the user can also change the filter for the task list to finished tasks and to all tasks. This way, the user could look at the details of completed tasks. Tapping any of the tasks in the list will open the task detail view (figure 4.5).



**Figure 4.4:** A list of maintenance tasks in the first version of the Boom Mobile app. Note that the height of a single entry is rather low, making it hard to select.

The user interface also contains a footer which shows the synchronization status. It shows a notification if the data is out of sync along with the time of the last synchronization. It also displays when the synchronization is currently in progress.



**Figure 4.5:** The detail view of a task

#### 4.4.2 Criticism of the prototype

On August 29 2011, a meeting of the developers from the external company, a Boom Software representative and the usability team was held. The prototype along with some design decisions leading up to the user interface was discussed. The following list contains some of the more noteworthy points that were discussed in detail (meeting and discussion at Boom Software's office, August 29, 2011):

- **Height of entries in the task list:** One major point of criticism from the usability team was the height of the entries in the task list (figure 4.4). On the target device, each entry had a height of 7.5 millimeters, which makes it hard to select a single entry. Consequently, the discussion moved to how many tasks

would typically be displayed in the list at once. According to Boom Software, this would be a small number, typically between three and five. The prototype however, has clearly been designed for larger lists. It was agreed upon that the item height would be increased (to roughly twice the current height).

Furthermore, it was discussed if it would be feasible to have multiple lines per list item in order to display additional details for each item. This would however cause problems with regard to the sorting of the items and the simple list layout, so the idea was dismissed.

- **Responsiveness:** There was a problem with the application being unresponsive after the login (after the user enters the password and presses enter / OK). The application would appear to be frozen and not respond to any input while also showing no progress bar or loading icon at all. This was probably due to the fact that the application needs to sync the UI with the server, which can take a little time. It was decided that this issue will be looked into and fixed until the next milestone.

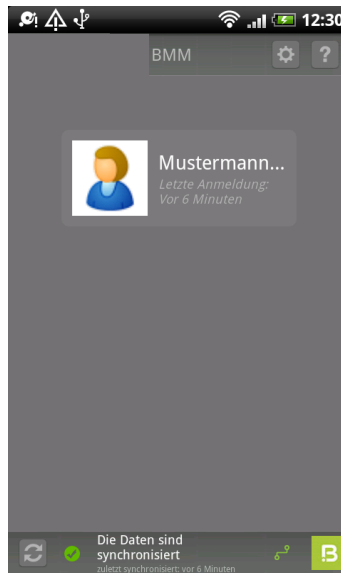
- **Displaying the Android status bar:** The discussion moved on to the question why the application was displayed in fullscreen mode, which makes the Android status bar disappear. Displaying the status bar is the default setting and generally considered best practice unless the designers want to make sure that the users' attention is not distracted at all, e.g. for games and video playback. The status bar shows the time and notifications such as received e-mails and text messages.

Everybody agreed that the status bar should be visible in the application and the change was implemented with the next milestone.

- **Bad error message:** An inexpressive error ("error 101") message was displayed upon startup. The reason for this was that the user's device has not been registered with its International Mobile Equipment Identity (IMEI) number with the server. The user should receive an appropriate message telling him to contact the administrator to fix the issue in this case.

### 4.4.3 Improved version

Many of the points that were criticised in the prototype application were fixed in the next version of the application. The following figures show the improved version and briefly describe some of the changes made:



**Figure 4.6:** The app now no longer hides the Android status bar (as can be seen at top of the image).

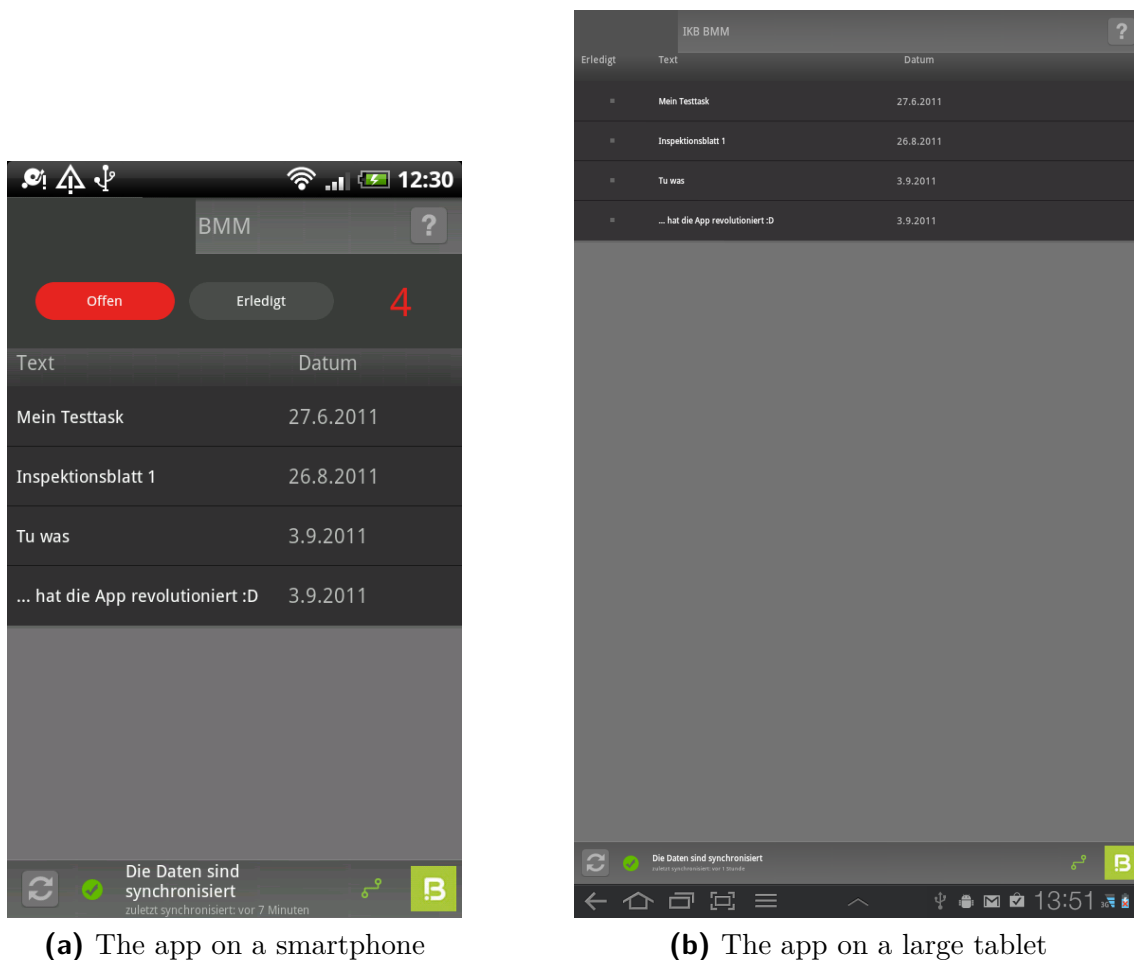


**Figure 4.7:** The task list in the improved version. List entries are now about twice as high.

#### 4.4.4 Scalability

The application was intended to run on 7 inch Samsung Galaxy Tab devices, which fall into the large specification of the Android framework. The app has also been tried out on two other types of devices: An HTC Desire smartphone (a medium sized device) and a Samsung Galaxy Tab 10.1 (an extra large device). The screenshots below show the comparison of several screens of the app on these two devices.

The app runs very smoothly across all devices tested. Scaling down to the smartphone size works very well, having no negative impact on usability. Scaling up to the larger tablet size works as well, but some screens seem a little bit empty. While the app certainly doesn't leave a bad impression on extra large devices, there is some unused space which could be put to use in order to improve usability a bit more.



**Figure 4.8:** A comparison of the task list on a smartphone and a large tablet.

Conclusively - based on the experience with the Boom Mobile app - it can be said that Android apps scale quite well without any effort specifically put into defining multiple user interfaces for devices of different sizes. There are, however, some caveats to this. Some factors have positively influenced the results with regard to scalability: Firstly, the app was initially designed for a 7 inch tablet, which is slightly above the middle of the range of device sizes. If it had been designed for very small or extra large screens, more effort would have been needed in order to achieve good results on the opposite end of the range. Secondly, the graphics used had high enough resolutions for them to scale nicely to a slightly smaller and larger size. Designing a user interface with small icons and then attempting to scale it to a larger display can make the graphics look pixelated or blurred.

Finally, it must be pointed out that the app has been tested on devices that don't have a big size difference to the target device: The Galaxy Tab has a display diagonal of 7 inches while the Galaxy Tab 10.1 has a diagonal of 10.1 inches (44% larger). The HTC desire has a diagonal of 3.7 inches ( 47% smaller), but the comparatively high resolution (800 x 480 pixels compared to 1024 x 600 pixels) made the perceived difference smaller than that. The app has not been tested on a very small device because there was none available (and testing on the Android emulator was impossible due to technical restrictions - devices need to provide a valid IMEI in order to start the app).

#### **4.4.5 Future development**

The development of the Boom Mobile app has not been finished yet. However, development of a Windows 7 version of the app (for Windows-based tablets) has been started as well. The Windows 7 version of the app will be developed by Boom Software. There were concerns regarding the acceptance of Android, which might require additional training of employees not yet familiar with the platform. Since there is a greater familiarity with Windows among the target audience, it has been decided to offer a Windows version as well in the future. Another advantage of the Windows 7 version that came up during the design phase is the fact that the login screen will not be needed as the user account inside the app can be directly linked

to the Windows user account (thus moving the login procedure to the startup of the device instead of the app).

The usability team might continue its role as advisors for the Windows 7 version, but any further development will not be included in this thesis.



## 4.5 Usability evaluation of desktop applications

In order to improve the usability of Boom Software's applications and to create UI design guidelines for the developers, usability evaluations of three representative existing applications - along with a few new features - were performed.

### 4.5.1 Heuristic Evaluation

A usability inspection using Heuristic evaluation was performed on three applications: The first application to be tested was the "Leserate", which is not an actual application which was shipped to customers, but subject of the tutorial of the BORA framework for new developers at Boom Software. It was considered important since it contains many typical UI elements and it is the first time new developers get in touch with the framework, so the tutorial application should display exemplary usability. For the heuristic evaluation, it also served as a pilot test for refining the test methods and getting feedback.

The other two application to be inspected were versions of the Boom Maintenance Manager (BMM) and the Boom Target Manager (BTM) in configurations as they were actually used by customers. The BTM is simply a rich-client desktop application while the BMM consist of a desktop client as well as a web interface for administrative tasks - both of which were tested.

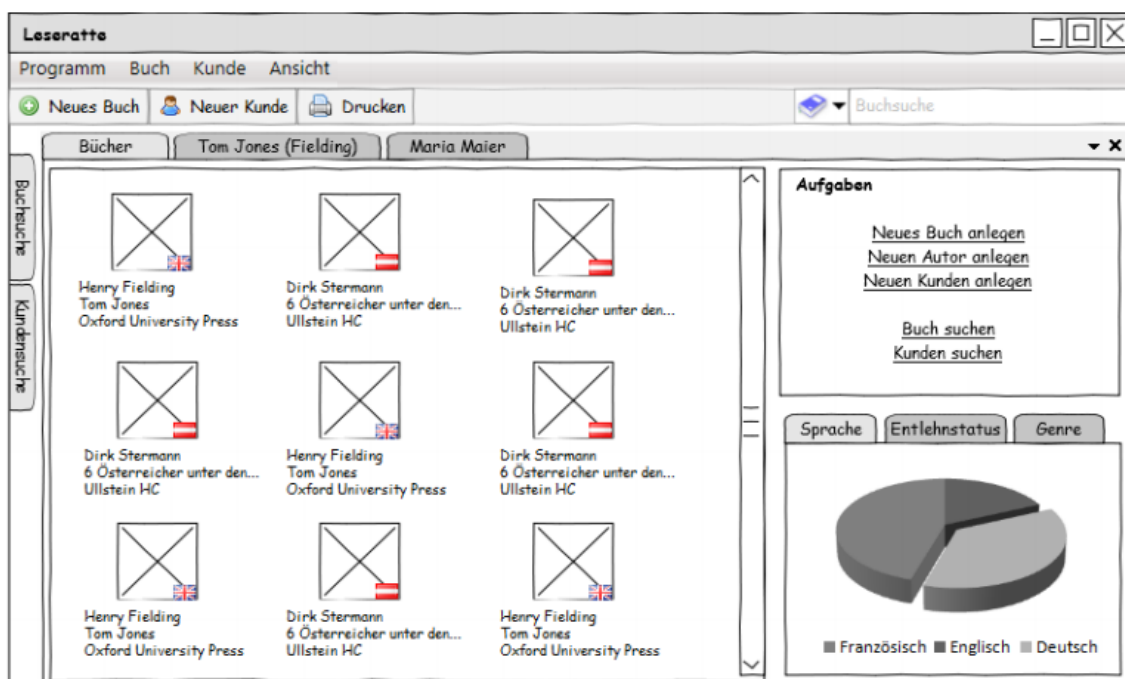
### 4.5.2 Thinking Aloud Tests

For the Thinking Aloud Tests the usability team created an improved version of the "Leserate". The improvements include fixes of bugs and flaws in the original version which were found in the heuristic evaluation (however, some were not fixed on purpose in order to see their impact in the TA tests) and extensions of the functionality. The functionality was aimed to be enough to be properly used in the test and to be credible in the given scenario. It also contained some features that Boom Software wanted to include in future versions of the BORA framework, but had not tested yet. The features and contents to be used in the improved Leserate were discussed and agreed upon by the usability team and Boom Software.

## The Leseratte application

The improved Leseratte application which was created specifically for the TA tests is a simple library management software. It enables library employees to add, modify or delete data on the books available. These data include book details, such as title, charge for borrowing the book per day or ISBN number, information on authors and publications. The application also allows the management of the library's customers as well as lending books to those customers.

The application is based on the developers' tutorial for Boom's BORA framework. It was created using the BORA framework, but it included components that were - at the point of creation - not fully integrated in the framework yet.



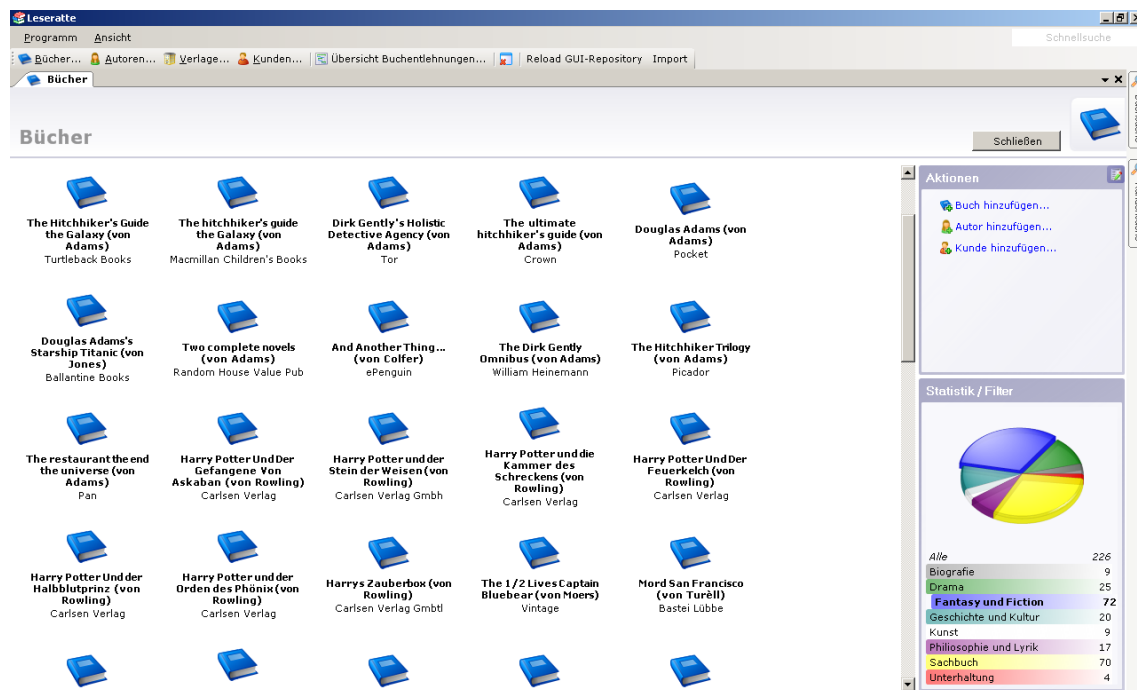
**Figure 4.9:** A mockup of the user interface for the improved Leseratte.

On startup the application presents the user with a start screen (see figure 4.10). The start screen shows all the books that are in the database, laid out in a grid view (similar to the "tile" setting in Microsoft's Windows Explorer). Each book is represented by a book icon (which is either blue or gray, depending on whether the book is available), its title followed by its author (in parentheses) and its publisher. The grid view was chosen on purpose in contrast to a table-like detailed list view in order to see the users' reactions to it.

The bottom right of the start screen contains the statistics / filter section, which

shows how many books there are by genre in the form of a pie chart as well as a list. It allows the user to filter the book list by (Ctrl-) clicking one or more of the genres. The selected genres are highlighted, as can be seen in figure 4.10 where the "fantasy and fiction" genre has been selected, as indicated by the elevated segment of the pie chart as well as the bold text.

The top right hand section of the start screen contains links to frequently needed tasks: Adding books, authors and customers.



**Figure 4.10:** The start screen of the Leseratte application. The Book list is currently open and books are filtered by the "fantasy and fiction" genre, as can be seen on the filter in the bottom right. Above the filter there are links to some of the most common tasks: Add book, add author and add customer.

To provide the test users with a sizable amount of data, a small automated helper program was written that queried the Google Books API <sup>1</sup> for books from a set of predefined authors. This way the program's database contained information on 226 books along with their authors, genres and publishers. The helper program also created a small number of predefined customers. While the amount of data is still significantly smaller than it would be in a real library, this gave the test users enough data to interact with through the course of the tests.

<sup>1</sup><http://code.google.com/apis/books/>, retrieved March 11, 2012

Leserate contains a number of other views, most importantly the list views for authors, customers and publishers. These can be accessed through the menu bar or through the toolbar below the menu bar (see figure 4.10). These views show tables of the respective items along with their attributes. By double-clicking any item, the detail view is opened. The detail view for a book can be seen in figure 4.11. Here the employees can edit the book details, look at some details that aren't shown in the list views and lend the book to a customer (or mark it as available when it has been returned). The detail views for authors, customers etc. look similar, albeit with fewer attributes.

The screenshot shows the 'Leserate' application window. The title bar reads 'Leserate'. The menu bar includes 'Programm' and 'Ansicht'. The toolbar contains icons for 'Bücher...', 'Autoren...', 'Verlage...', 'Kunden...', 'Übersicht Buchentlehnungen...', 'Reload GUI-Repository', and 'Import'. The main window title is 'The Hitchhiker's Guide to the Galaxy (von Adams)'. Below the title are buttons for 'Speichern', 'Abbrechen', and 'Kopieren'. The main content area is divided into two columns:

- Kopfdaten:**
  - Buchtitel: The Hitchhiker's Guide to the Galaxy
  - Autoren: Douglas Adams
  - Verlag: Pan
  - Buchformat: Einband: Taschenbuch
  - Genre: Fantasy und Fiction
- Weitere Daten:**
  - ISBN-10: 0330513087
  - ISBN-13: 9780330513081
  - Zusammenfassung / Klappentext:
  - Schlüsselwörter:
  - Buchsprache: Englisch
  - Seiten: 0
  - Veröffentlichungsdatum: 01.09.2009
  - Preis pro Tag: 0,42
  - Preis pro Woche: 2,94

At the bottom left, there is a section titled 'Entlehnstatus' with buttons for 'Entleihen' and 'Zurückgeben'.

**Figure 4.11:** The detailed view on a book. In this form, the employees can change any data on the book through the form elements or lend it to a customer by clicking the button in the bottom left.

Another noteworthy view which was implemented for the test is a calendar view which shows a list of all currently borrowed books, along with the corresponding customer and the return date. The application also includes a search for books and customers, which can be accessed through a flap on the right side of the screen. All attributes of books (e.g. author, genre, publication date) and customers can be searched.

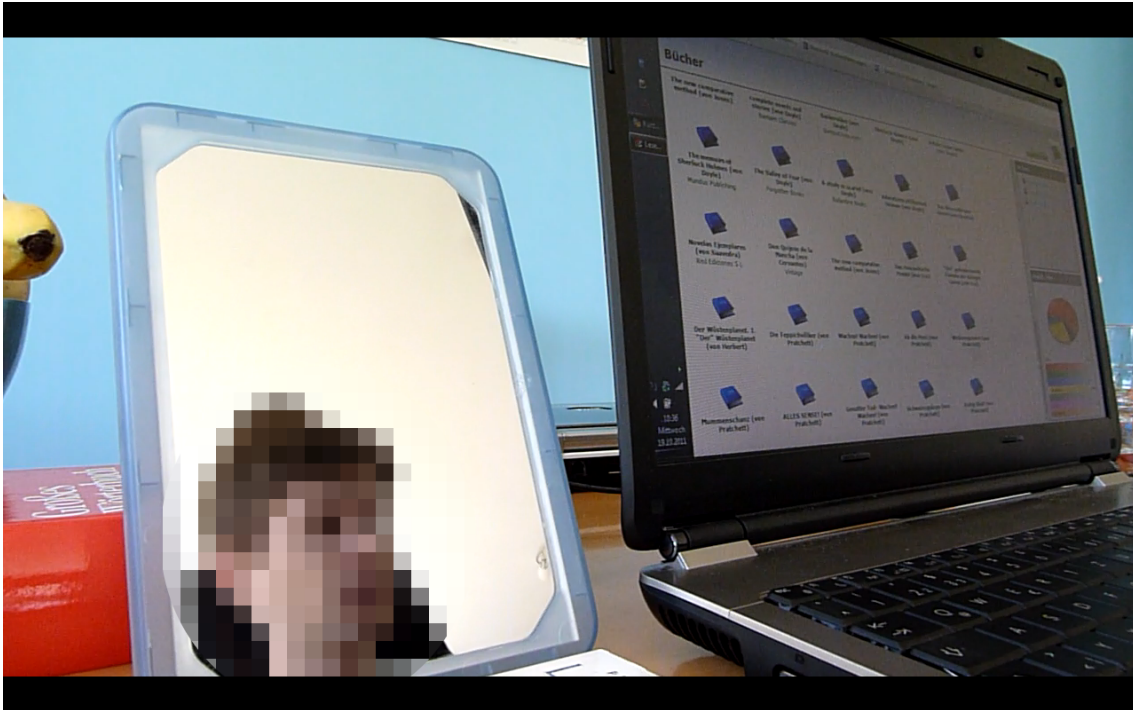
## Test setup

A total of nine test users were testing the software. The test with the first user was intended to be a pilot test in order to check the difficulty of the individual tasks and refine the testing procedure. However, only minimal adaptations were made, so the results from the pilot test could be used and evaluated along with the other test results.

Originally, the team intended to have five test users plus one pilot test user, following the findings of Nielsen (1991), who argues that TA test results were very similar for five test users and for ten test users and concludes that no more than five test users are needed. Boom Software requested more test users in order to have more extensive findings, so the team agreed on eight users.

The test users were between 22 and 54 years old. Three of them were male and six were female. They had varying computer literacy, ranging from moderately experienced to experts ("power users"). A "bottom-line" level of moderate computer literacy was set in order to match the target audience of Boom Software's actual products - The test users had to use computer programs on a regular basis and have a couple of years of experience with their usage.

The tests were conducted in different locations. Some were done in the apartments of the usability team's members while some tests were done at different locations, such as Boom Software's office. The locations were chosen so that the testers would not have to travel too far and so that a quiet room was available for the tests.



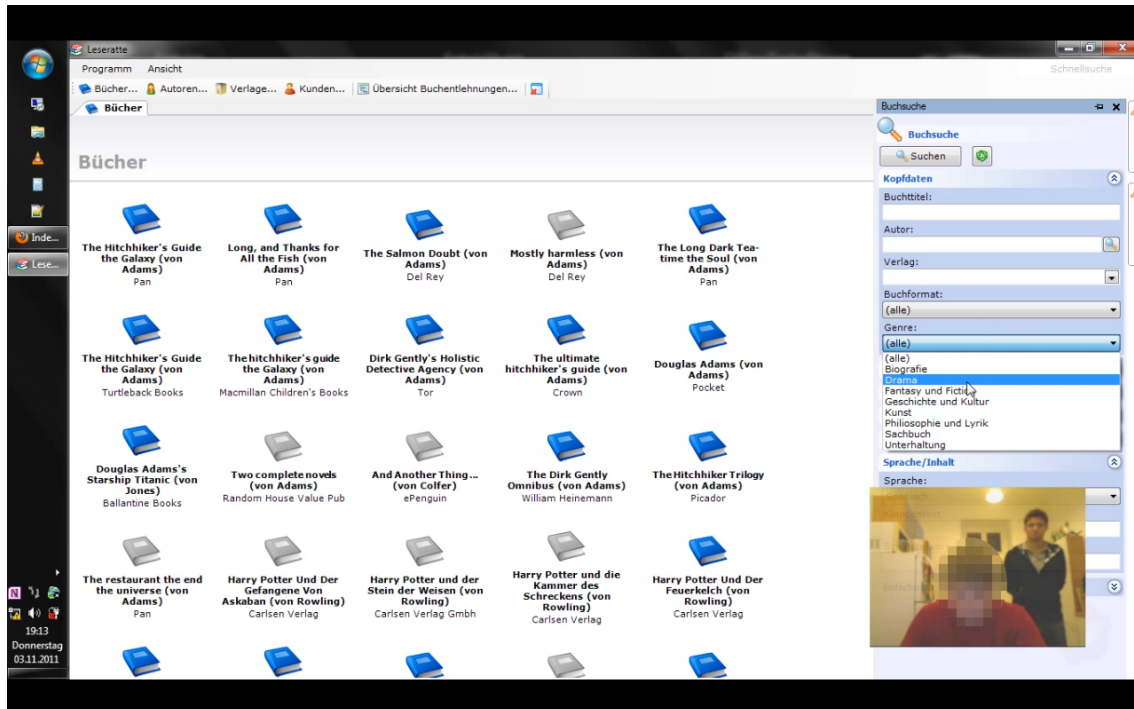
**Figure 4.12:** The setup of a thinking aloud test. The test person (pixelated for privacy reasons) is sitting in front of a laptop and using the Leseratte application. This recording comes from the backup camera. The test person is also being recorded by the web cam and the screen contents are being captured. One member of the team assumes the role of the test leader, giving the tasks to the test person. The other team member takes notes of noteworthy events.

A laptop running Windows 7 was used for the tests. The detailed specifications can be found in table 4.1.

The tests were recorded using a Logitech ®USB webcam, which was mounted on the laptop screen. Audio was recorded from the laptop's internal microphone. The screen contents were captured using the software Morae by TechSmith. Additionally, a camcorder and a tripod were used to provide a backup recording. The test users were filmed from diagonally behind with the camera focused on the screen. A mirror was placed next to the laptop screen so that the user's facial expressions could be analyzed (as can be seen in figure 4.12).

Display	15.6" widescreen (16:9)
Resolution	1366 x 768 pixels
CPU	Intel®Core™i5-560M
RAM	8 GB
Operating System	Windows 7 Service Pack 1
Tested software	Leseratte 0.9.0.0 (build from Oct. 27th, 2011)

**Table 4.1:** Hardware and software specifications of the laptop used for the thinking aloud tests



**Figure 4.13:** A screenshot of a TA test recording made with Morae. The contents of the screen have been recorded and are shown in full size. The video recording of the test user can be seen in the bottom right.

## Test procedure

The test procedure for every single test was as follows:

1. Greeting.
2. Short explanation of the test, its purpose and the testing procedure.
3. Request the test user to read the introduction document explaining the test.  
Discuss any open questions with the test user.
4. Prompt the user to sign the NDA and the declaration of consent.

5. Hand the the user the background questionnaire and let him fill it out.
6. Check if computer settings (mouse speed, display brightness etc.) are comfortable for the user
7. Start video and audio recording (screen capturing and camcorder)
8. User attempts to complete the tasks of the actual TA test
9. Conduct interview
10. Prompt the user to fill out a feedback form
11. Stop recording.

### **Test tasks**

The test users were given the following tasks (with approximate time limits in parentheses). The first task was an introductory task which served for the users to get comfortable with speaking out their thoughts aloud. The tasks were meant to guide the users through most of the main features of the application, in some cases leaving them a choice on how to approach a problem.

1. Find out today's cinema program at 8pm in a given cinema in town (2 minutes)
2. Gather impressions, get to know the application, get an overview of its features (3 minutes)
3. Find a given book and give detailed information about it (3 minutes)
4. Create a new customer from given data. (2 minutes)
5. Filtering and searching
  - (a) Filtering books by genre and authors by first letter of last name (5 minutes)
  - (b) Search for books by given specific criteria (5 minutes)
6. Lend a book to a customer (3 minutes)
7. Find out return dates on the calendar (5 minutes)



8. Add a new book, including as many details as possible - actual book was given to test users (10 minutes)

Completion criteria for every task were established. If the user failed to meet these criteria within the time limit the task was skipped. However, the time limits were not set in stone and the team used good judgment in borderline cases, granting the test users a little extra time if they were about to finish the current task slightly outside the time limit.

### **The post-test interview**

An interview was conducted with every test user after they finished the test. The interviews were started by asking the test users how they liked the test, which alone often yielded extensive answers. The usability team then asked for positive and negative impressions, attempting to get a clear picture of what the user liked and disliked while avoiding leading questions of any kind. If any questions or events worth discussing emerged during the test they were discussed as well. For instance, if a user could not complete a task, he was shown how it could have been completed. Then the team and the user discussed why the test user had not found the solution and how the program could be improved to make it easier for users to solve the problem in question.

## **4.6 Usability guidelines**

There were two final products derived from the usability inspection and consulting: Firstly, an improvement of the BORA framework in general in order to make it easier for the developers to ensure good usability in applications. Secondly, a set of usability guidelines was established in the company. These are aimed at the developers and designers and should offer a simple guidance for UI design and for usability decisions.

It is important to raise the awareness of usability and usability issues among developers and UI designers. According to Nielsen (1991), people are better at finding usability problems in user interfaces when they know more about usability principles.

Severity rating	Meaning
4	Catastrophic problem
3	Serious problem
2	Minor problem
1	Cosmetic problem
0	Not a problem

**Table 4.2:** Severity ratings used for problems identified in the Heuristic Evaluation and the Thinking Aloud Test.

The main source for these guidelines were, of course, the results from the usability tests. Problems identified either by the test users (throughout the test or the interview) or by the usability team were weighted by severity and by the count of occurrences. The severity ratings were adopted from Nielsen (1994) and are explained in table 4.2.

The usability guidelines are - as of February 2012 - not finished yet. A subset of the guidelines is currently undergoing internal revision before officially being made available to the developers.

## 5. Results

### 5.1 Usability evaluation results

This section outlines the main findings of the heuristic evaluation and the thinking aloud tests.

#### 5.1.1 Heuristic evaluation

The findings of the two usability team members have been combined. Severity ratings have been added to every issue identified. The issues have then been listed sorted by severity, but separately for the three UIs under test. Although there weren't any issues that were exactly the same across all UIs, some definitely fall into the same category, giving valuable information on parts of the framework or general areas that need improvement. A report on positive and negative impressions on all programs was composed as well. The most important findings will be presented here.

#### **Main findings**

The programs tested were overall mostly well usable. The consistency, the flexible window / tab system and the form layout overall were rated as strong points of the UI (albeit with some exceptions). Some more detailed positive examples include the auto-complete features of some form elements or the highlighting of the related inputs when there were errors.

Common negative observations were (among others) problems with feedback (mostly bad error messages and the lack of a progress indicator for unresponsive tasks), some problems and inconsistencies with form elements and a few tasks that seemed un-

necessarily complicated.

Out of the three programs tested, Leseratte was by far the least complex. Consequently, it was also the easiest to use and had a very clear UI. There were some significant problems as well, but many were actually bugs which could be traced back to errors in the tutorial. Other findings suggest that significant improvements could be made with regard to simplicity and ease of use, but that is understandable since the program tested was only meant to serve as a tutorial to developers.

With the Boom Maintenance Manager, there were different results for the web application and the desktop client. The web application contained the most issues of all the UIs that were tested, giving it a rather flawed overall impression. The main points of criticism address the lack of overview and clarity of the UI. The overall structure of the application seems to be unnecessarily complicated. Furthermore, the user feedback was in need of improvement - error and information messages were often unclear.


The BMM desktop client on the other hand had the fewest issues of all UIs. Despite the complexity of the underlying business processes it was easy to use and had a clear UI.

The Boom Target Manager seemed mature overall. The most significant negative impressions were related to performance issues, with some waiting times without any kind of feedback or status information. Many of the other issues were related to individual controls or views of the UI, which were not clearly labeled or explained. The limit of only 50 search results being displayed simultaneously was a hindrance for some tasks. Furthermore, there were minor problems with feedback to the user and the highlighting or emphasizing of controls.

### **Sample findings**

A total of 105 issues were found in the heuristic evaluation of the four user interfaces. One example is illustrated in figure 5.1. The problem identified is "inconsistent alignment of labels (right- vs. left-aligned)". There is a description which gives more details on the problem and the place where the issue occurs or the way

to reproduce it. The corresponding heuristic is also given - in this case Nielsen's heuristic number four: Consistency and standards. A screenshot was provided to illustrate the problem at a glance where applicable. In the example the problem quickly becomes evident.

Nr.	4
Titel	Inkonsistente Ausrichtung der Labels (rechts vs. linksbündig)
Beschreibung	Im Formular zum Hinzufügen / Bearbeiten von Büchern sind die Labels teilweise rechts- und teilweise linksbündig.
Ort	Book → Neu → Kopfdaten
Heuristiken	N4 - Consistency and standards
Screenshot	

**Figure 5.1:** A sample issue which was identified during the heuristic evaluation of the Leseratte program.

The above example illustrates a very specific and a minor error. Another issue found in the HE of the Leseratte - to also list a different example - was the start screen, which gave few hints to the functions of the program. No window is open by default and all the user sees is the menu bar along with the search window. While the menu does lead to the core functionalities of the program, they could have been made much easier accessible and more visible.

### 5.1.2 Thinking aloud test

The videos taken for the thinking aloud test were thoroughly analyzed. Every notable event was marked with a time stamp and a type (e.g. positive impression, problem) and commented. The complete event log was then examined and similar events were grouped together. The resulting positive impressions and negative impressions / problems were described in detail (including references to the points in the videos when they occurred and screenshots). The final report of the TA test was 52 pages long and shall not be discussed here in detail. Instead, the main findings will be presented along with some selected problems and positive impressions.

## Main findings

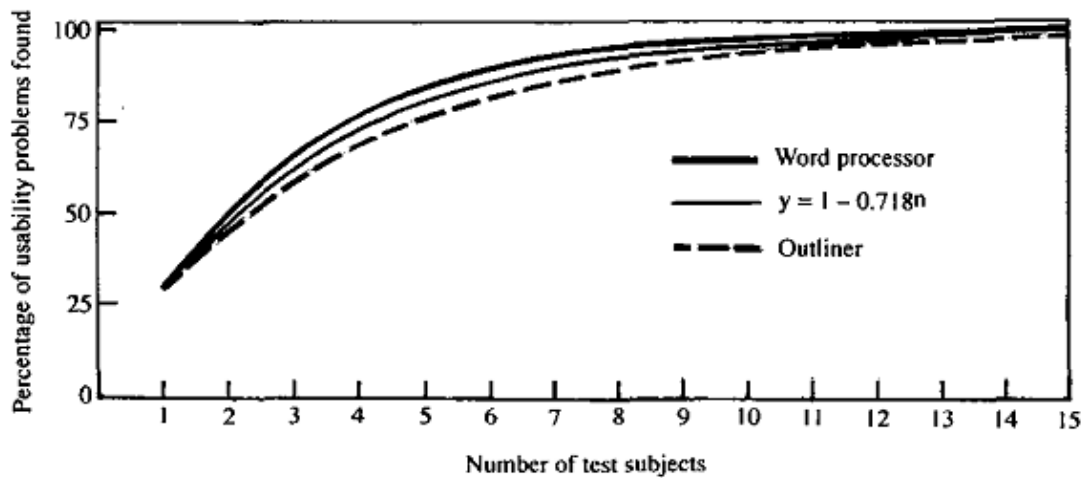
The Leseratte application was overall well-received by the test users. Among the positive impressions mentioned were the simple and clear overall structure of the user interface and the easy navigation. The book search was generally deemed well usable and useful, with many users mentioning the detailed search options.

There were, however, also problems that were identified through the TA test. The tasks for filtering and for using the calendar view seemed to be especially challenging for the test users, suggesting the need for improvement. These problems were caused by the lack of visibility of the user interface elements used for filtering and by the generally not easily understandable calendar view respectively.

Beside the problems observed during the tests, the users' criticism included the lack of filtering options in the calendar and book views. Also, the laborious adding of other data (e.g. a new author) while adding a new book was a common problem. Furthermore, some elements of the UI were not clearly labelled and had no help information whatsoever. Some features were not used by many users, either because they simply did not see them or because they found another way to accomplish their tasks. This might in part be due to the task formulations, which, while trying to be general and leave the choice on the approach to the user, still necessarily emphasize certain elements of the UI while at the same time putting others in the background. Many of the flaws that were identified were estimated to be repairable with low to moderate effort. The usability team is confident that with some updates to the framework and the introduction of the usability guidelines, the usability of future programs will improve.

When examining the number of users in retrospect and trying to measure the additional information gain with nine instead of the intended five to six test users, it can be said that every new test user did bring some additional insights. However, the amount of new information decreased with every new user. The first five test users already brought up most of the more severe issues, with the following users mostly adding details to the existing issues or adding minor issues. Although no empirical scores on issues found per user count have been created, it can be estimated that the progression roughly matches that of Nielsen (1994).

In the end, choosing the number of test users comes down to weighing the costs and benefits of a higher test user count.



**Figure 5.2:** Usability problems found with number of test users, source: Nielsen (1994)

### Positive impressions

The positive impressions that the test users talked about were usually very general in their nature. They commended the overall ease of use of the program or features (such as the book search) as a whole. They indicate that the overall impressions of most sections or features of the program were good. However, during the interviews after the tests many times the comments were phrased in the fashion of "in general I like that, but...", with the users criticising details despite the good general impression.

There were also exceptions to this observation, namely some users positively commenting on details such as the input fields for dates and the tooltips of individual components.

All positive impressions have been documented. An example can be seen in figure 5.3.

### Negative impressions / problems

The variety of problems that were found is great. Some were critical and definitely warrant quick improvement while others were minor cosmetic issues or not actually

**P1 Übersichtlichkeit und Einfachheit des Programms**

TPs	TP0 (0:00:28, Video 2; 0:05:30, Video 2), TP1 (Fragebogen, Interview), TP2 (0:26:55, 0:27:17), TP3 (0:32:00), TP4 (Interview), TP5 (0:26:35, 0:42:08), TP6 (0:22:18), TP7 (0:33:02), TP8 (0:30:12)
Beschreibung:	Übersichtlichkeit und Einfachheit des Programms
Zitate:	0:00:28 (TP0, Video 2): Test als einfach empfunden (=> Bedienung des Programms einfach) TP0 (0:05:30, Video 2): "Oben Ansichten Bücher etc., rechts oben konkrete Aktionen. "Habe ich gleich verstanden"" TP1 (Fragebogen): "Wenn man es ein paar Mal übt, ist es sicher leicht." TP1 (Interview): "Das Programm ist sehr übersichtlich - auf einen Blick ist alles zu sehen."

**Figure 5.3:** An excerpt of a positive impression which was shared by many of the test users. The caption reads "Clearness and simplicity of the program". The entry includes the references to the test videos (including a timestamp), a description and quotes by the users. In this instance, test user 1 stated that "The program is very clear - everything can be seen at a glance".

usability problems, but rather feature requests. Figure 5.4 shows the description of a problem which was rated a minor problem (2 out of 4 points) by the usability team.

Out of the 41 problems identified, four were rated as critical (average score of 3.5 or greater) and eleven more were rated as serious (score of 2.5 or greater).


The users also offered some suggestions for improvements on some occasions. These suggestions have also been documented. They mostly relate directly to the problems mentioned. The quite numerous expression of explicit suggestions (17 in total) for improvement implies that the users do not only identify usability flaws, they also have clear expectations on how things should look instead. Most of the suggestions were basis, straightforward improvements, but they are nevertheless helpful, as they show where actual users - in contrast to UI designers - would expect improvements.

## 5.2 Usability guidelines

After the usability team had categorized, rated and prioritized all the findings, usability guidelines were developed by Boom Software. This section discusses the



#### N19 Zurücksetzen-Button bei der Suche schwer zu erkennen

Severity:	2
TPs	TP2 (0:13:16), TP3 (0:16:31), TP4 (0:09:09), TP6 (0:10:29), TP8 (0:11:18)
Beschreibung:	Der Zurücksetzen-Button wurde von den meisten Testusern nicht erkannt. In ein paar Fällen wurde die Bedeutung des Buttons erst durch den Mouse-over Tooltip erkannt, in den meisten jedoch gar nicht - Die gesetzten Felder wurden einzeln geändert oder gelöscht.
Zitate:	TP2 (0:13:16): "Da muss man alles wieder zurücksetzen" (Feldinhalte wurden einzeln gelöscht) TP2 (Interview): "Der Zurücksetzen-Button sieht eher aus wie "Nochmal Suchen" oder so."
Screenshot:	

**Figure 5.4:** An example of an identified problem with the program. As illustrated in the screenshot, the "reset" button in the book search form has a misleading icon. Most test users didn't guess its meaning from looking at it, with some mistaking it for a refresh button. The button did offer a tooltip description when the user hovered over it with the mouse cursor, but that might not be discovered.

initial version of the guidelines, which were internally published in late January 2012 and still undergoing revision. The guidelines are accessible through Boom Software's web portal, which allows access only to authenticated users. The majority of framework- and design-related issues have been addressed in the guidelines already.

The guidelines contain two sections: General UI design guidelines and framework-related guidelines.

### 5.2.1 General UI design guidelines

The general guidelines section contains background material that the usability team has used throughout the tests and general information that came up during the research for the theses.

The first part of the section lists and explains some usability heuristics and rules. The list includes the ten usability heuristics by Nielsen (1993), Shneiderman's "8

Golden Rules of Interface Design" (Shneiderman et al. (2009)) and the Rules of Usability by Raymond and Landley<sup>1</sup>.

Furthermore, the general guidelines include a set of users' behavioral patterns. It is based on the behavioral patterns described by Tidwell (2011), with a focus on those patterns that are relevant to Boom Software's target audience. Knowledge of the patterns should help designers and developers to understand the users and employ user-centered design. Also, since the user interfaces of Boom Softwares' programs often use a lot of forms, many guidelines and best practices on UI forms and controls as described by Tidwell (2011) were also included.

When working with a platform such as Microsoft Windows, it is important to follow established standards. For this reason, the guidelines also include a reference to Microsoft's Windows User Experience Interaction Guidelines<sup>2</sup>. Some of the conventions mentioned there, such as dialog layout, button order and naming conventions, should be known and used by designers.

---

<sup>1</sup><http://catb.org/~esr/writings/taouu/html/ch01s03.html>, retrieved Feb. 7, 2012

<sup>2</sup><http://msdn.microsoft.com/en-us/library/windows/desktop/aa511440.aspx>, retrieved Feb. 8, 2012

## 5.2.2 Framework-specific guidelines

The BORA framework specific section of the guidelines builds upon the results of the heuristic evaluation and the thinking aloud tests, while also keeping in mind the conventions established by existing Boom Software products. The framework-specific guidelines contain multiple chapters, ranging from general rules to a detailed checklist to be used on a completely designed user interface.

Again, a representative example of the numerous guidelines shall be given. On the subject of the main window / start window, the guidelines say:

### **The main functions of the program should be evident at the first glance**

A user's first look at a program is crucial. Within the first few seconds after starting the program, the user tries to find out what purpose the program should serve, what kinds of information and which functions are relevant.

Therefore, the start screen or the initially visible windows and menus should be both tidy (i.e. not overloaded) and filled exactly with those commands and informations that are most important for the users to get started with their work smoothly. Peripheral or unimportant features and features aimed at powerusers can be placed in submenus or extensible tool windows.

Example: If the application is used for task management, the menu items for creating and opening tasks should be placed most prominently. The most important informations for the user might include the recently edited tasks, open tasks assigned to him or notifications from the system. It can be a good idea to provide a

dedicated start screen, i.e. a document window which is initially opened on startup in order to allow the user to get started with the program. The start screen should be clearly laid out and might include some of the following functions / areas:

- Personal notifications
- Recently used / modified object (e.g. tasks, persons etc.)
- A search function ("Google"-like)
- Upcoming appointments and open tasks
- Statistics / state of the system (e.g. number of open tasks, number of users logged in, version number)
- An area with links to the main functions (e.g. create task, search / print bill)

All these informations need not be displayed in a complete fashion. It suffices if they are hinted at (e.g. not all open tasks, only the next upcoming five) and serve as an entry point or navigational method to more detailed views.

It is imperative that a good amount of core functions can be accessed through the main window's toolbar. This is described in detail in the guidelines' section for rules for menus.

The above guideline was adapted directly from the results of the usability tests, which stated that the initially opened windows of the programs often left the users confused as to what the core functions were and where to find them.

The guideline gives the designers some general hints as to how the start screen of an application should look like, thus helping to create consistent user interfaces.

## 6. Discussion and Lessons Learned

This chapter attempts to summarize the key findings from both the scientific research and the practical project and discuss some of the most important aspects.

### 6.1 Mobile UI design

A broad range of research was performed on the subject of UI design on various mobile platforms. When looking at individual platforms, the results vary. Different platforms have different ranges of screen sizes available. The Android platform is the most segmented in this regard, but it also offers a wide array of tools for designing UIs for multiple screens, such as density independent pixels and size standards. One shortcoming of the Android platform is the lack of support for scalable vector graphics. Therefore, designers must not only ensure that the layout looks good on all target devices, but also that graphics are available in the correct resolutions.

Android allows the distinction and definition of UIs for different screen size purely through XML layout definitions. Therefore, the UI definition is clearly separated from the rest of the code base, thus encouraging proper use of the Model View Controller (MVC) pattern.

The iOS platform handles the different resolutions of older and newer devices quite well by measuring the resolution in the abstract unit of points rather than physical pixels. The same UI layout can therefore be used for old and new iPhones and iPod Touch devices with different screen resolutions.

The iPad, however, differs from the other iOS devices in size, resolution and aspect ratio. It is therefore recommended to design separate UIs for iPad and iPhone / iPod Touch.

With Windows Phone 7, the issue of scaling has been neglected so far, as only devices with a resolution of 800 x 480 pixels are available on the market. It will be interesting to see whether additional resolutions will be permitted in the future and, if so, how the resulting problems will be handled.

Conclusively, following platform conventions is very important on mobile devices, especially when developing native apps. This has been part of the heuristic "Consistency and standards" by Nielsen (1994), but usability experts need to be aware of the platform-specific guidelines for all target platform in order to successfully apply this heuristic to mobile apps.

The MVC pattern can be put to use very efficiently on mobile platforms and with HTML5. All platforms offer easy means to separate the MVC components. However, they do not force adherence to the MVC pattern, which means that is ultimately the responsibility of the developers and designers to properly use it.

Using the MVC pattern for multi-platform apps is definitely beneficial, because the definition and layout of the UI are isolated from the rest of the code base and can therefore easily be modified for multiple target devices.

### **6.1.1 Choosing a method for app development**

When comparing native apps to web apps in general, there is no clear recommendation to be given. The choice of whether to develop native apps, web apps, hybrid apps or to use a multi-platform framework depends on a number of things. The requirements of the app and the resources available as well as the target audience should be considered. The skills of the developers are another important factor, as learning how to develop apps for one or more mobile platforms can be very time-consuming. On the other hand, getting to know the APIs of multi-platform frameworks can also take some time, but it will most likely be less than what it takes to learn multiple native APIs.

Having developers who are already skilled in one or more platforms' native programming languages and APIs or in HTML5, CSS and JavaScript can be a great asset

for app development which should be leveraged if possible.

It is also worth mentioning that multi-platform frameworks are growing very rapidly and both updates to existing frameworks and the introduction of new frameworks occur frequently. We recommend to compare the features of the most recent versions of various frameworks before deciding on one.

Another very important thing is to be aware of the limitations of the frameworks. First and foremost, these framework only offer tools for creating UIs for multiple platforms and screen sizes. It is the responsibility of the designers and the developers to ensure that the apps run properly on all the target platforms and devices. None of the frameworks can substitute thorough testing of the app.

## **6.2 Usability inspection and testing**

Both the heuristic evaluation and the thinking aloud tests gave the team a lot of results and feedback to work with. Some of the issues identified were easy to fix and fixing them made the applications under test more usable with very little effort. The tested applications left positive overall impressions on the team and the test users, also pointing out some of the strengths of the applications and the framework in general. The large number of findings (both positive and negative in nature) and the resulting benefits clearly underline the usefulness and importance of performing usability evaluations.

There have been some redundancies from performing both a heuristic evaluation and a large number of thinking aloud tests on applications that are quite similar regarding their user interfaces. These redundancies were, however, quite small. Only very few issues have surfaced in both the heuristic evaluations and the thinking aloud tests while a number of positive impressions have been identified by both. It can be argued that this is due to the fact that positive observations - whether they come from usability experts or average test users - tend to be more general in nature while negative impressions and problems are often related to specific elements or parts of the interface, which can be pinpointed by the test users or evaluators.

Among others, the clearness and simplicity of the UIs and the consistency have been observed in both the HE and the TA tests. While the issues identified are mostly program-specific, some are related or can be put into the same categories, such as UI elements which are unclear due to the lack of explanations (e.g. tooltips) or tasks which could be simplified (reducing the number of clicks / actions required for the users).

Conclusively, performing both heuristic evaluations and thinking aloud tests has been beneficial for the project and the resulting usability guidelines.

The number of TA test users has been briefly discussed before. It is always hard to recommend a number of test users for upcoming tests. While the three to five users recommended by Nielsen (1994) can be confirmed to be a good general rule of thumb, we encourage usability evaluators to carefully consider additional factors, such as the complexity of the software, the number of tasks given to the test users and the temporal and financial budget for the tests, and make a more sophisticated choice for the number of test users.

Another approach we can recommend would be to perform additional tests as long as enough new insights are gained with every new test (as similarly suggested by Holzinger (2006)). This approach requires quite a bit of flexibility as the number of test users can only be roughly estimated beforehand and finding additional test users and scheduling tests takes some time. The tests and their analyses would also have to be done back-to-back, because analysis is needed in order to decide whether or not to run additional tests.

When looking at the differences between the two evaluation methods (see figure 2.5), Holzinger (2005) suggests that the two methods are aimed at different stages of the development process: Heuristic evaluation can be used on any stage and thinking aloud tests are best suited for the design stage. The usability team performed the thinking aloud tests with software which was created specifically for the tests. Conceptionally, the software was situated somewhere between the design phase (testing new UI elements) and a running system (tried and tested elements), while to the test users the software appeared to be a finished and running product. While testing a (more or less) finished product has no negative impact on the knowl-



edge gained, it is most often harder to fix the issues, as large-scale modifications of existing software can be expensive. In the case of Boom Software, changes to the framework or to core modules are very hard to make, as a large number of end products would be affected. Some of the issues will therefore not be retroactively fixed in the existing applications, but rather implemented in future releases.

When evaluators are aware of these implications as well as the effort and time required, performing TA tests on already released software should prove no problem.

### **6.3 Usability guidelines**

The guidelines developed by Boom Software and the usability team have only been made available to developers recently (with an official presentation on Tuesday, March 6th 2012), so no statements can be made concerning their effectiveness and usefulness. The reception from the developers who attended the presentation was positive overall and the usability team could convey the importance of following the established usability guidelines.

One important aspect of the guidelines is that they contain both practical experience (from the usability evaluations) and scientific information (as researched by the team members). While the findings have, in many cases, been specific, it is important not to force them upon the developers. The usability team has performed the usability evaluations and done the background research for the guidelines, but it was left to Boom Software to develop guidelines from these findings on their own. This participation in creating the guidelines will hopefully underline the fact that they come from a collaboration between Boom Software and the usability team, rather than being solely based on external scientific research.

The usability team is, of course, aware that the guidelines in their current form are by no means final. The team encourages internal evaluation and collaboration in order to refine or adapt the guidelines however Boom Software may see fit. Future input for the guidelines may come from experiences of the developers, feedback from customers or possible future usability evaluations.

Conclusively, the work with Boom Software has been profitable for both sides. While

Boom Software could evaluate their products and establish usability guidelines, the usability team could add real-life, industry experience and thus a bigger engineering aspect to their scientific work.

## 6.4 The Boom Mobile app

While the development of the Boom Mobile app has unfortunately not been completed as of the writing of this thesis, the short period in which the usability team has provided usability consulting for the app has yielded some interesting results.

A brief evaluation of an early prototype has uncovered a number of usability issues which were, at this stage of development, still easy to fix. This clearly speaks in favor of rapid prototyping and early usability evaluation.

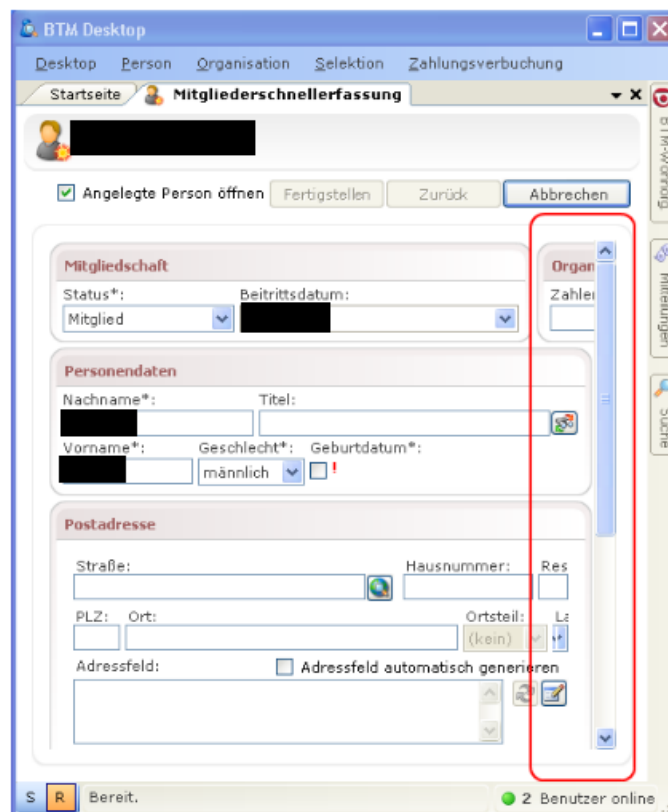
One of the major issues was the height of list entries. The real problem behind this issue was a misconception regarding the amount of tasks of actual users. Luckily this issue was found early in the development process. Applying user-centered design from the very beginning of the design process could have made this issue clear even earlier. In this case, the developers used randomly created test data, which didn't match the data actual end users would have used. Therefore we strongly recommend to put effort into the construction of meaningful, relevant test data.

## 6.5 Scalability of Boom Software's UIs

The scalability of the Boom Mobile app has already been discussed. There were also some issues related to scalability in Boom Software's desktop applications. Figure 6.1 shows an issue where shrinking the window created a vertical scrollbar, but no horizontal scrollbar. In the situation depicted, the right part of the form is inaccessible (by means other than enlarging the window).

While Boom Software has specified a display resolution of 1024 x 768 pixels as a design guideline, scalability is still important. Firstly, the BORA Framework allows users to rearrange windows in applications, which can effectively make them much smaller than that. Secondly, with the planned advance into the mobile market, smaller resolutions and screen sizes should be considered as early as possible.

The issue was acknowledged by Boom Software and a related guideline was created.



**Figure 6.1:** An issue found in the Boom Maintenance Manager which is related to scalability. There is no horizontal scrollbar, making the right part of the form inaccessible in a small window. Personal data has been censored.



## 7. Conclusions

Research has shown that the different mobile platforms offer different tools and means to scale user interfaces. The Android platform is the most fragmented one with regard to the range of screen sizes, but it also offers the most tools for designers. Multi-platform frameworks can also offer some assistance with the scaling of user interfaces. These are, however, only tools and it is up to the designers and the developers to ensure that their user interfaces look good and are well usable on all target devices. While the well thought-out definition of layouts is a good foundation for this, it must be stressed that there is no way around testing the interface. This can be done on actual devices or on various configurations of the emulators provided by the platform IDEs.

The model-view-controller pattern was encountered on all platforms and also in multi-platform development. Most frameworks build, in some way, upon this pattern. Designers and developers should follow the conventions set by these frameworks and be sure to properly separate the components of the MVC pattern. By doing so, only the views need to be modified for different devices while the rest of the code can remain unchanged.

The practical work in cooperation with Boom Software has shown that an Android app, which was developed specifically for 7 inch tablets, scales well and is also usable on smaller smartphones and larger tablets. On closer examination, however, it becomes evident that there is still room for improvements and optimizations, especially on larger screens.

The usability evaluation performed has underlined the importance of usability inspection and testing. By evaluating actual large-scale software we could gain a great amount of insights. Many positive impressions and many issues came up, along with

suggestions on how to fix them. We are looking forward to seeing the results of our efforts in future products of Boom Software.

During a workshop held at Boom Software's office we could observe the impact of the awareness for usability guidelines and issues firsthand. After the workshop, developers and designers could single-handedly identify several usability issues in their software. We therefore strongly recommend trainings on usability in order to raise awareness among any software development team and encourage the emphasis on usability across all stages of development.

When inspecting mobile user interfaces, all the established heuristics for desktop applications still hold true. There are some additional issues which apply specifically for mobile devices or for touch input. The usability evaluators need to know these issues and should also have some experience with the target platforms. Since there is no definitive list of rules or checklist with regard to current mobile devices, evaluators should be well aware of the target platforms' guidelines.

Another important point, which surfaced in both theoretical research and the usability evaluations, was user-centered design. Especially with mobile apps it is crucial to be aware of the users' needs and goals. When prototyping and testing, meaningful test data should be used, as randomly generated test data can lead to misconceptions.

With regard to multi-platform development and the choice between developing native apps or HTML5 web apps, no definitive recommendation can be given. The best choice always depends on a number of factors, such as the app's intended features, the target audience or the skills of the development team.

Multi-platform frameworks offer some interesting features, especially for rapid prototyping or the development of very simple apps. These frameworks are growing rapidly and new features are being added frequently, so we recommend to inspect several platforms' features before starting multi-platform development and carefully choosing a well-suited framework.

## 8. Future Work

This thesis has covered a broad range of topics and some of them clearly warrant further research.

While this thesis has addressed many of the design guidelines of individual mobile platforms it would be interesting to combine and summarize these guidelines into one set of guidelines for multi-platform mobile UI design. These could include both the specific recommendations from the platform guidelines and general established usability standards. Of course, the guidelines would have to be evaluated, for instance by using them in an actual heuristic evaluation.

One of the major points in the conclusion of this thesis was that mobile interfaces need to be tested on all target devices. While tools for automated UI tests exist (e.g. Robotium for Android<sup>1</sup>), these are mostly aimed at functional tests (i.e. whether a certain user input yields the correct output). The research and development of a tool which automatically inspects a mobile UI for one or more platforms and informs the designers about potential issues would be an interesting continuation of this thought.

With regard to the usability project with Boom Software the initial usability evaluation and the creation of the guidelines are finished. We recommend to monitor the ongoing use and improvement of the usability guidelines. Furthermore, additional evaluation of Boom Software's products should be performed at regular intervals in the future in order to measure the success of the improvements and the guidelines. The Boom Mobile app is also still under development and usability consulting and evaluation of the app could yield interesting insights for research, should Boom Software be interested in further cooperation.

---

<sup>1</sup><http://code.google.com/p/robotium/>, retrieved March 11, 2012

Multi-platform development frameworks are also an area with room for further research. In this thesis a few frameworks were briefly examined. A suggestion for future work would be an extensive test of multiple frameworks. A multi-platform app (which has its features clearly specified) could be implemented by different developers, each using a different framework. The results could then be thoroughly compared on multiple devices.



## List of Figures

2.1	Smartphone and PC shipments for 2011 . . . . .	21
2.2	Smartphone shipments for 2011 by platform . . . . .	22
2.3	Control flow of the MVC pattern . . . . .	24
2.4	Back-of-device interaction on very small devices . . . . .	25
2.5	Comparison of various usability evaluation methods . . . . .	28
3.1	Share of various screen sizes across the Android platform . . . . .	35
3.2	Definition of nine-patch graphics . . . . .	38
3.3	An example of nine-patch graphics . . . . .	38
3.4	Illustration of Android's back stack . . . . .	39
3.5	Example of Android fragments on a smartphone and a tablet . . . . .	40
3.6	A simple example of the action bar pattern . . . . .	41
3.7	The Windows Phone 7 home screen . . . . .	42
3.8	Windows Phone 7 display resolutions . . . . .	44
3.9	Size comparison of iOS hardware . . . . .	46
3.10	The view hierarchy of a sample iOS application . . . . .	47
3.11	The storyboard of an iOS app . . . . .	47
3.12	The Codiqa rapid prototyping tool for jQuery Mobile . . . . .	55
4.1	The simplified BORA process model . . . . .	61
4.2	The core components of the BORA framework . . . . .	62
4.3	The login screen of the Boom Mobile app. . . . .	63
4.4	The task list in the Boom Mobile app . . . . .	64

4.5	The detail view in the Boom Mobile app . . . . .	65
4.6	Improved version of the Boom Mobile app's login screen . . . . .	67
4.7	Improved version of the Boom Mobile app's task list . . . . .	67
4.8	A comparison of the task list on a smartphone and a large tablet. . .	68
4.9	A mockup of the user interface for the improved Leseratte. . . . .	72
4.10	The start screen of the improved Leseratte application . . . . .	73
4.11	The detailed view on a book in Leseratte . . . . .	74
4.12	The setup of a thinking aloud test . . . . .	76
4.13	A screenshot of a TA test recording . . . . .	77
5.1	A sample issue which was identified during the heuristic evaluation of the Leseratte program. . . . .	83
5.2	Usability problems found with number of test users . . . . .	85
5.3	An excerpt of a positive impression from the TA tests . . . . .	86
5.4	An example of a problem identified in the TA tests . . . . .	87
6.1	A scalability issue identified in the HE . . . . .	97

## List of Tables

3.1	Android screen size standards . . . . .	34
3.2	Android screen diagonals . . . . .	34
3.3	Android screen resolutions . . . . .	34
3.4	Share of various screen sizes across the Android platform . . . . .	35
3.5	Comparison of the screen sizes of iOS devices . . . . .	45
3.6	Screen sizes of iOS devices in points . . . . .	48
3.7	Programming languages used for app development on various mobile platforms . . . . .	51
3.8	Advantages of native and HTML5 web apps . . . . .	52
4.1	Hardware and software specifications of the laptop used for the thinking aloud tests . . . . .	77
4.2	Severity ratings used for problems identified in the Heuristic Evaluation and the Thinking Aloud Test. . . . .	80



## References

- Baudisch, Patrick and Gerry Chu [2009]. *Back-of-Device Interaction Allows Creating Very Small Touch Devices*. *Interface*.
- Baudisch, Patrick M., G.F. Petschnigg, D.H. Wykes, A.Y.S. Shum, Avi Geiger, K.P. Hinckley, M.J. Sinclair, J.B. Jacobs, J.D. Friedman, R.H. Ho, and Others [2008]. *TRACKING INPUT IN A SCREEN-REFLECTIVE INTERFACE ENVIRONMENT*. <http://www.google.com/patents?hl=en&lr=&vid=USPATAPP12175695&id=9aHLAAAAEBAJ&oi=fnd&dq=Tracking+input+in+a+screen-reflective+interface+environment&printsec=abstract>.
- Bevan, Nigel [1995]. *Measuring usability as quality of use*. *Software Quality Journal*, 4(2), pages 115–130. <http://www.springerlink.com/index/G744753360415047.pdf>.
- (Boom Software AG) [2012]. *About Boom*. <http://www.boomsoftware.com/en/about-us>.
- Card, SK and TP Moran [1980]. *The keystroke-level model for user performance time with interactive systems*. *Communications of the ACM*. <http://dl.acm.org/citation.cfm?id=358895>.
- Chae, Minhee and Jinwoo Kim [2004]. *Do size and structure matter to mobile users? An empirical study of the effects of screen size, information structure, and task complexity on user activities with standard web phones*. *Behaviour & Information Technology*, 23(3), pages 165–181. ISSN 0144-929X. doi: 10.1080/01449290410001669923. <http://www.informaworld.com/openurl?>

genre=article&doi=10.1080/01449290410001669923&magic=crossref|  
|D404A21C5BB053405B1A640AFFD44AE3.

Freeman, E., B. Bates, and K. Sierra [2004]. *Head first design patterns*, volume 1. O'Reilly & Associates, Inc. <http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstracthttp://dl.acm.org/citation.cfm?id=1076324>.

Holz, Christian and Patrick Baudisch [2011]. *Understanding touch. Proceedings of the 2011 annual conference on Human factors in computing systems - CHI '11*, page 2501. doi:10.1145/1978942.1979308. <http://portal.acm.org/citation.cfm?doid=1978942.1979308>.

Holzinger, A., K.H. Struggl, and M. Debevc [2010]. *Applying Model-View-Controller (MVC) in design and development of information systems: An example of smart assistive script breakdown in an e-Business application. e-Business (ICE-B), Proceedings of the 2010 International Conference on e-Business*, pages 1–6. [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=5740449](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5740449).

Holzinger, Andreas [2005]. *Usability Engineering Methods for Software Developers. Communications of the ACM*, 48(1), pages 71–74.

Holzinger, Andreas [2006]. *Thinking-aloud eine Königsmethode im Usability Engineering. Informatik Akademie der Österreichischen Computer Gesellschaft*, (1977), pages 4–5.

Leske, Nicola and David Cowell [2011]. *Android conquers almost 50 percent of smartphone market*. <http://www.reuters.com/article/2011/08/01/us-smartphones-research-idUSTRE7704MS20110801>.

Meier, Reto (Google) and Max (Google) Mahemoff [2011]. *HTML5 versus Android : Apps or Web for Mobile Development?*

Microsoft []. *Hardware Specifications for Windows Phone*. [http://msdn.microsoft.com/en-us/library/ff637514\(v=vs.92\).aspx](http://msdn.microsoft.com/en-us/library/ff637514(v=vs.92).aspx).

Nielsen, Jakob [1991]. *Trip Report : Usability Metrics and Methodologies. SIGCHI Bulletin*, 23(2), pages 107–108.

- Nielsen, Jakob [1993]. *Usability Engineering*. Morgan Kaufmann. ISBN 0125184069, 362 pages. <http://www.amazon.com/Usability-Engineering-Jakob-Nielsen/dp/0125184069>.
- Nielsen, Jakob [1994]. *Estimating the number of subjects needed for a thinking aloud test*.
- Petzold, Charles [2010]. *Programming Windows Phone 7*. Microsoft Press. ISBN 978-0-7356-4335-2, 1082 pages.
- Schmidt, Albrecht [2000]. *Implicit human computer interaction through context. Personal Technologies*, 4(2-3), pages 191–199. ISSN 0949-2054. doi:10.1007/BF01324126. <http://www.springerlink.com/index/10.1007/BF01324126>.
- Sears, Andrew and Ben Shneiderman [1991]. *High Precision Touchscreens : Design Strategies and Comparisons with a Mouse. International Journal of Man-Machine Studies*, 34(4), pages 593 – 613.
- Shneiderman, Ben, Catherine Plaisant, Maxine Cohen, and Steven Jacobs [2009]. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition)*. Addison Wesley. ISBN 0321537351, 624 pages. <http://www.amazon.com/Designing-User-Interface-Human-Computer-Interaction/dp/0321537351>.
- Siek, Katie A, Yvonne Rogers, and Kay H Connelly [2005]. *Fat Finger Worries : How Older and Younger Users Physically Interact with PDAs. Ifip International Federation For Information Processing*, pages 267–280.
- Tidwell, Jenifer [2011]. *Designing Interfaces*. O’Reilly Media. ISBN 1449379702, 576 pages. <http://www.amazon.com/Designing-Interfaces-Jenifer-Tidwell/dp/1449379702>.
- Zalewski, G.M. and C. Nicholson [2009]. *HAND-HELD DEVICE WITH TWO-FINGER TOUCH TRIGGERED SELECTION AND TRANSFORMATION OF ACTIVE ELEMENTS*. <http://www.google.com/patents?hl=en&lr=&vid=USPATAPP12574860&id=B7PdAAAAEBAJ&oi=fnd&dq=Hand-Held+device+with+two-finger+touch+triggered+selection+and+transformation+of+active+elements&printsec=abstract>.