Automated and Unsupervised User Interaction Logging as Basis for Usability Evaluation of Mobile Applications

Florian Lettner
Department for Mobile Computing
University of Applied Sciences Upper Austria
Softwarepark 11, 4232 Hagenberg, Austria
florian.lettner@fh-hagenberg.at

Clemens Holzmann
Department for Mobile Computing
University of Applied Sciences Upper Austria
Softwarepark 11, 4232 Hagenberg, Austria
clemens.holzmann@fh-hagenberg.at

ABSTRACT

The evaluation of mobile user interfaces can be a tedious task, especially if usability tests under real-world conditions should be performed. In particular, the evaluation of high-fidelity prototypes provides valuable measures about the quality of mobile applications, which helps designers to identify potentials of improvement for the next revision. Due to their costs or missing expert knowledge evaluation techniques such as cognitive walkthroughs or heuristic evaluation are often not used. Additionally, commercial frameworks provide insufficient details on usability as they only address commercial statistics regarding user loyalty, in-app purchases or demographics.

In this paper, we present a novel approach and toolkit for automated and unsupervised evaluation of mobile applications that, in contrast to existing frameworks, is able to trace any user interaction during the entire lifecycle of an application. As a major novelty, our toolkit can be added to mobile applications without changing application source code, which makes it flexible and scalable for all types of applications. Also, our toolkit is able to identify and visualize design flaws such as navigational errors or efficiency for mobile applications.

Keywords

mobile interfaces, usability metrics, software toolkit

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Evaluation/methodology*

General Terms

Measurement, Performance, Human Factors

MoMM2012, 3-5 December, 2012, Bali, Indonesia.

1. INTRODUCTION

Nowadays, the mobile market is saturated with a huge amount of applications. As a consequence, consumers are not able to decide which application to buy solely based on functional features. If consumers have to choose between two applications that basically provide the same functionality, they will prefer the application that presents this functionality to them in a comprehensive manner so they can benefit most from it [12].

As a result, effective and easy user interfaces are crucial for a mobile application to be successful. Moreover, compared to desktop applications or web sites, mobile applications have to cope with external stimuli, as application users might not be sitting in front of a big screen for substantial amounts of time [11]. Instead, it is more likely that they will walk down the street or sit in a train when they use a mobile application. Hence, it is important not to ignore the differences between desktop and mobile usability and monitor mobile usability under real-world conditions and not in isolated usability laboratories without any distraction.

Therefore, we first identify the challenges for automatic usability evaluation of mobile applications in the next subsection, such as measuring usability under real-life conditions in the mobile space. We analyse prevalent approaches for supervised and unsupervised field studies, and demonstrate in which terms our approach differs from the related work.

Secondly, we give an overview on the contributions of this paper. We outline how we can achieve good results for mobile usability evaluation with our novel software toolkit that is not limited to laboratory-based usability testing.

1.1 Problem Description

Although most of the usability methodologies (e.g. usability inspection, heuristics, etc.) are both applicable to desktop as well as to mobile applications, it is more difficult for mobile applications to achieve statistically significant and at the same time practically relevant results with these conventional evaluation methods. The reason is that the emulation of real-world use during a laboratory-based evaluation is only feasible for a precisely defined user context. Thus, due to a lack of different contexts and physical limitations, it is difficult to generalize from quickly changing and possibly strongly varying user context [17].

Although Nielsen shows that field studies carried out in the post-release phase reveal usability issues that might result in fatal consequences [16], Kaikkonen et al. point out that field studies are not the best solution for the evaluation of usability on mobile devices [6]. They assume that field studies require twice the effort and time than laboratory studies (i.e. preparation, supervision of test participants, etc.). They also point out that running pre-tests or pilots is critical based on the assumption that there are many

things that can go wrong, and that usability testers have to ensure that everything works out correctly during the field study.

Apart from conventional usability evaluation methodologies there are some commercial frameworks for logging user statistics on mobile devices, such as Flurry[1],Google Analytics[2], Localytics[3] or UserMetrix[4]. However, these frameworks focus on descriptive user statistics such as user growth, demographics and commercial metrics like in-app purchases. They are able to identify if an application pleases its customers or not, but they cannot identify the reason behind this observation. According to Tullis et al. [20], these user metrics are important, but insufficient data to gain a deeper insight in application usability. As a consequence, although these frameworks are able to track basic user statistics, they cannot explicitly provide data to identify usability metrics such as efficiency, user errors or user satisfaction.

1.2 Contributions

Basically, good user interface design can make the difference between product acceptance and rejection in the marketplace. In contrast to Kaikkonen et al. [6], we believe that conducting unsupervised and automatic field studies will result in more thorough results than it is the case for supervised field studies. This assumption is supported by Hertzum [4], who shows that the results of field studies differ between *workshop tests* and *laboratory-based tests*. Hertzum showed that conducting unsupervised field studies is cheap and does not require much preparation. Moreover, more representative data is provided by leaving almost everything to the users, as more users have access to the study on the one hand. On the other hand these users are not guided by supervisors who could influence the study by foreclosing interaction possibilities.

Thus, we present a methodology and a working prototype of a toolkit consisting of a mobile framework, an IDE wizard and a cloud-based web server, which allows developers to perform usability-related experiments in the field. With the presented approach we are able to transparently log user interaction data and compute usability-related metrics and descriptive statistics data during unsupervised field studies from unaware end-users. This data is then used to present the results to developers automatically. On the one hand, the presented toolkit is capable of collecting metrics such as simple counts (i.e. hit rates, hit/miss ratios, time on screen) but on the other hand these count metrics are used in order to create completely new and intuitive visualisations to display the users' workflow throughout entire mobile applications. In addition to logging user navigation between different screens, we also provide a methodology to log on-screen interaction. This enables developers to understand the user interaction with UI components such as buttons or text fields as well as the usage of multi-touch gestures.

With the proposed toolkit, we are able to present statistics where the user interaction is in focus. In contrast to the frameworks mentioned in the previous section, we can show *why* users maybe stop using an application and not just that they do. The second major contribution is that, thanks to our adoption of aspect-oriented programming which will be explained later, developers do not require insight into the internal functionality of our toolkit. They can deploy it out of the box using the provided IDE wizard without having to manually identify cutting points (i.e. points where interaction events have to be captured) and insert framework code.

Thus, we are able to overcome the problems mentioned in the previous section as on the one hand developers do not have to add framework code manually to their applications – this makes the integration process cheap, very easy, effective and robust against human errors – and on the other hand user interaction data can be collected transparently for unsupervised field studies. This provides developers with the possibility to perform fast and effective unsupervised field studies with end-users without the need for extensive preparations.

1.3 Outline

The contributions of this paper, as outlined above, are twofold. First, we give an introduction on the developed toolkit and how it can be used in order to perform field tests under real-life conditions. This part of the paper outlines the basic infrastructure in which field tests can be performed and how test data can be collected and analysed.

Second, we identify usability metrics that are suitable for the mobile context. According to Tullis et al. [20] choosing the right statistics for usability evaluation is critical. They present different types of statistical procedures to collect and evaluate nominal, ordinal, interval and ratio data. Within this paper, we refer to these metrics as low-level metrics, and they provide raw and statistically analysed data.

Based on the results gained from a conducted feasibility study with about 300 users, we will show that it is possible to construct usability metrics on top of these low-level metrics, which allow for assessing the quality of mobile application interfaces. In order to verify and to validate our approach, we conducted a feasibility study with an application that can be downloaded from the Android Play Store.

2. RELATED WORK

Although usability evaluation is a very important aspect for application development, mobile application design and usability measurement is in its infancy. Moreover, usability engineering for mobile applications is done in a quite platform-specific way. For example, Jakob Nielsen generally sees in mobile usability evaluation more the evaluation of mobile websites (i.e. optimised websites for mobile phone browsers) and not necessarily the evaluation of mobile applications, which are installed natively on the phones. However, Wilson et al. [22] point out that there is a lack of tool support for automatic usability evaluation. In the following, several projects that contribute to the work within this paper are described and related to our approach.

2.1 EvaHelper Framework

In 2009, Balagtas-Fernandez et al. [1] presented an Android-based methodology and framework to simplify usability analysis on mobile devices. The EvaHelper Framework is a 4-ary logging system that records usability metrics based on a model presented by Zhang et al. [1]. Although Balagtas-Fernandez et al. focus on the part of automatically logging user interaction, they do not focus on the evaluation of the collected data. For the visualization of their results, they use third-party graph frameworks that are based on GraphML[5] to visualize a user's navigational graph. Compared to our approach, they solely provide navigational data for single application usages. However, they do not augment the graph with some kind of low-level metrics to enable the identification of navigational errors or of unused components.

2.2 RobotME

[1] http://www.flurry.com
[2] http://www.google.com/intl/de/analytics/
[3] http://www.localytics.com/
[4] http://usermetrix.com/
[5] http://graphml.graphdrawing.org

In 2007, Marcin Zduniak [24] presented a framework for the automated GUI-testing of JavaME applications. The framework of Zduniak is, like our approach, capable of automatically recording usability data during application usage in the field. However, Zduniak does not take any metrics into account. His framework is built for recording and replaying user input, which allows to simulate user interaction with form-based applications only but not for analyzing and evaluating the recorded data. In general the RobotME project is more suitable for context analysis, as Zduniak's solution focuses on the collection of user data which does not directly relate to usability. Technically speaking, our approach is similar to Zduniak's. However, it is not restricted to JavaME. Moreover, we do not collect user input in terms of personal data (e.g. chat messages, content of forms, stored application data), but we focus on data that is directly linked with usability metrics. In this context, we are able to capture any user interaction automatically, which involves user navigation between different screens, but also interaction within single screens. Thereby, we are able to identify frequently used view components (e.g. buttons, text fields, etc.). Unlike Zduniak, who focuses on subjective input data that addresses a single user, we are focusing on data that describes objectively *how an application is used*. As a result, in addition to data provided by single users, we are able to aggregate interaction data for an arbitrarily large amount of users.

2.3 Flurry Analytics

Commercial frameworks such as Flurry[6], which is taken as a representative for commercially available analytic frameworks (i.e. Localytics[7], Mobilytics[8], Appuware[9] and UserMetrix[10]), try to get a deep audience insight. They provide usage statistics based on metrics like average users per day, new users per week or user loyalty. In order to use these frameworks, the development teams offer support and code snippets to integrate the framework with existing applications. However, developers are responsible for adding framework functionality at the right place in their applications. Besides the option of collecting demographic information (i.e. gender, age or location), the framework also offers a possibility to track custom events. However, metrics that provide assumptions about the quality of user interfaces or the lifecycle of applications are generally missing.

In contrast to these frameworks, our approach directly focuses on the quality and structure of user interfaces. It takes data collected by third-party frameworks (i.e. age, gender or location) into account to filter or group usability data to ensure statistically significant and not random results.

2.4 Discussion

Basically, most of the automated approaches for usability evaluation focus only on the collection of raw statistical data. However, they either do not take into account the interpretation of the collected data, or they do not provide options for visual representations of the results. Table 1 lists the different approaches and compares them according to the provided functionality.

Basically, the EvaHelper project is a good approach for a logging system. The data is recorded semi-automatically, which means that the developers are responsible for adding the correct function calls to their applications. However, this approach – and the RobotME project as well – does not provide options for the interpretation and visualization of the collected data. In contrast to our approach, the visualization for the related work only uses graph-based libraries to visualize the user's navigation, but metrics are completely ignored. Moreover, RobotME is solely based on the collection of user input. In contrast, users can provide user specific data based on their own decisions for our approach (e.g. age, gender, experience). This can be combined with our future work, where we try to derive user satisfaction from electronic questionnaires, which is currently only available for cognitive walkthroughs. However, cognitive walkthroughs are not automatic methodologies, but are related to common laboratory-based usability evaluation.

[6]http://www.flurry.com/
[7]http://www.localytics.com/
[8]http://www.mobilytics.net/
[9]http://www.appuware.com/
[10]http://usermetrix.com/

EH = EvaHelper RM = RobotME
FL = Flurry AD = (An)Druid

Criteria	EH	RM	FL	AD
Automatic Data Collection	*o*	√	χ	√
Portability	√	χ	*o*	√
Visualization	*o*	χ	√	√
GUI Evaluation	*o*	χ	χ	√
User Input	χ	√	*o*	*o*
Navigational Burden	*o*	χ	χ	√
Task Definition	χ	χ	*o*	*o*
User Satisfaction	χ	χ	χ	√

Table 1: Effectiveness of related frameworks and applications

3. IMPLEMENTING THE PROPOSED TOOLKIT

Due to the lack of tool-based usability evaluation methodologies [22], especially for mobile applications, we decided to develop a toolkit to contribute to automatic and unsupervised usability evaluation. Our goal was to develop a software that can monitor interactions with any mobile interface, and which can provide preprocessed and interpreted statistics related to mobile usability data to aid developers and designers improving their user interfaces for mobile applications.

As a starting point, we evaluated commercially available frameworks (i.e. Flurry[1], Google Analytics[2], Localytics[3] or UserMetrix[4]) and found out that these frameworks do record user statistics and metrics that partly could be used for usability evaluation, but in terms of Nielsen and Shneiderman the collected data is more about learning something about the users than learning something about the quality of the user interface [15, 19]. Besides, in order to make use of these frameworks, developers have to change the source code of their applications manually by adding framework code. This also means that developers have to spend a lot of time in studying the framework architecture and functionality, which can cause high costs for any production team.

In contrast to these commercial frameworks, we want to provide a solution allowing both application designers and developers to get information on the quality of mobile applications beyond user statistics such as navigational behaviour and errors, design flaws, efficiency and user satisfaction. Additionally, we want to provide a toolkit that can be used out of the box and where developers do not have to change their applications by adding framework source code that makes application source code more difficult to read and prone to human errors.

Figure 1 shows the basic infrastructure of the presented toolkit, which was implemented for the Android platform during the first development iteration. Mobile application developers create applications with this toolkit, and they can be published in the Google

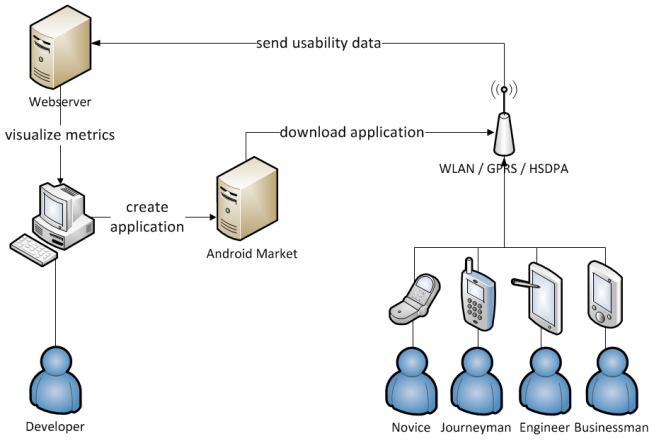

Figure 1: Basic infrastructure for the proposed toolkit.

Play Store afterwards. When users download such an application from the Play Store and work with it, the mobile framework transparently collects usability-related data and sends it to a web server. The web server finally pre-processes and interprets the collected data and visualises the usability metrics for the developer or application designer.

3.1 Requirements

One of the most important identified requirements was to provide the option of collecting usability-related statistics (i.e. low-level metrics and user behaviour) in the field. The logging process works totally transparently, which means that the user's workflow is not affected by the mobile framework. In other words, the framework does not require direct interaction with the user. Thus, the user is not distracted by the framework while operating with the host application. Thus, users are able to work with their mobile applications naturally and under real-life conditions. These requirements go along with the results of Hertzum's [4] analysis regarding field studies, where he suggests to not supervise test participants during the field study.

Basically, the toolkit does not foreclose the option for performing laboratory-based tests, where e.g. users have to follow a cognitive walkthrough. Usability testers can define user tasks as they would do for cognitive walkthroughs, but they do not need to manually collect test data during the test, as the toolkit handles the collection and interpretation (e.g. data aggregation, statistical validation, etc.) of the data internally.

In order to not violate the users' privacy, we based our toolkit on a opt-in strategy. This means that the first time an application equipped with the proposed framework launches, users can decide whether they want to activate it or not. Any application can be used even if the users do not activate the toolkit. Moreover, the collected data is not associated to personal information. Information sent to the evaluation server – such as age, gender or technical experience – have to be provided by users explicitly and voluntarily. Additional voluntary data can be used for filtering usability data on the server. This allows us to compare different user groups like for instance left-handed to right-handed users, male and female users, experienced to inexperienced users, etc.

3.2 Technical Premises

As shown in figure 2, the toolkit can be split into three major parts: (i) the mobile framework which is based on Android, (ii) the server backend based on the Google App Engine and (iii) the server frontend based on the Google Web Toolkit.

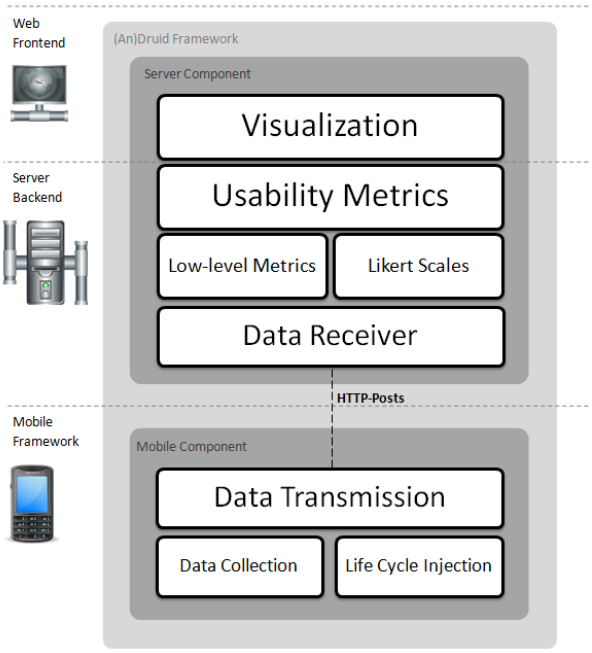

Figure 2: Illustrates the basic architecture of the proposed framework.

One of the main reasons why Kaikkonen et al. consider field studies not suitable for mobile usability evaluation in [6] is the assumption that the preparation and realization of field tests is too expensive and takes too much time. Therefore, we decided to use aspect-oriented programming (AOP) [7] for the implementation of the mobile framework. As a consequence, developers do not have to change application code. The configuration can be done using a wizard which is directly integrated in the IDE. A second compiler merges the mobile framework code and application code subsequently after it has been compiled. Thus, changes in the life cycle of an application (e.g. creating, showing, hiding a screen, interacting with buttons, text-fields, etc.) can be automatically identified, because each view container (e.g. screen, panel, etc.) as well as each visual component (e.g. button, text field, etc.) has its own life cycle consisting of a bundle of methods that are called sequentially. The collected data is transmitted to the server each time the user quits the application or – if it is desired – after a certain period of time or each day at a certain time. If data cannot be transmitted immediately, it is temporarily stored in the phone's cache and re-transmitted as soon as the phone's connectivity has been re-established.

The servlet-based server backend receives data sent by mobile clients. It constructs the navigation graph, calculates low-level metrics according to specifications provided by Tullis et al. [20] and saves this data to a schemaless NoSQL datastore, which is able to store Java beans as Java Data Objects (JDO). In order to present usability metrics such as efficiency, learnability, memorability, errors and user satisfaction to developers, the low-level metrics have to be organized and merged.

These usability metrics are visualized using a web frontend that is based on the Google Web Toolkit. It enables widget based presentation of charting tools and statistical plots using JavaScript, AJAX and HTML5 for an interactive and comfortable navigation experience.

4. AUTOMATIC TESTING WITH USABILITY METRICS

As we focus on collecting and providing statistics for unsupervised usability evaluation, our approach has to overcome some difficulties. First, we cannot provide the opportunity for the creation of user tasks, which would be required for cognitive walkthroughs [18]. However, it is not necessarily required to define specific application tasks except when they are connected to a certain context [2], if mobile applications only provide a single or at least a limited set of functions [11]. For instance, Google shows that by limiting the functionality for applications, users can understand the utility of each application easier. An example is the Google Maps mobile application, which can not be used for navigation; this function is provided in a different application. Finding points of interest is also not included in the Google Maps mobile application, for which users have to start the Google Places app.

Second, a way of measuring usability is needed. We use metrics for this purpose, which are a way of measuring or evaluating particular things in mathematical terms [20]. In terms of this paper, metrics provide measures for analysing and comparing user interfaces. Usability metrics have to be observable and quantifiable to convert them into a numeric format. Additionally, Nielsen identifies multiple benefits resulting from usability metrics that can be applied to desktop, web as well as mobile applications [14]:

- **Progress Tracking.** Metrics allow developers and designers to measure progress between different releases. Progress tracking allows a fine-tuning of the interface design process and helps controlling the application flow.
- **Competitive Comparison.** Metrics provide developers and designers with the possibility of comparing their interfaces to previous versions of the same applications or to competitive solutions available on the market.
- **Bonus Plans.** From a psychological point of view, metrics can help creating bonus plans for project members in order to enhance the productivity and to reward someone's engagement.

This section of the paper focuses on the identification of metrics for usability evaluation for single-function or limited-function applications according to the definitions of Madrigal et al. [11]. First, we will present low-level metrics that can easily be recorded using counts and ratios. We show which metrics are used within the proposed usability evaluation framework, how they are computed and how they are visualized.

Second, we will show how some of these low-level metrics can be used in order to derive usability metrics in order to measure effectiveness, learnability, memorability and user satisfaction.

4.1 Low-level Metrics

During the development of the proposed usability framework, we analysed the architecture of mobile applications for the Android platform. However, other platforms like Windows Phone have a similar structure, and our mobile framework is currently being implemented for them as well. We found out that they can be compared to web applications in terms of structure and navigation components. An Android application consists of different screens named *Activities*, which are connected via so-called *Intents*. An Intent basically is a hyperlink that changes the view from one Activity to another. Like for websites, it is possible to detect Activity changes due to callback methods that are part of each Android application (see Figure 3). These callback methods are called every time an intent, which is comparable to a hyperlink, is called to leave the current screen and navigate to a different one. Thus, the methods *onCreate()* and *onDestroy()* indicate when a new screen is opened or when it is closed, respectively. Additionally, *onResume()* and *onPause()* are used to record session times and they also count how often a user returns to a screen before leaving it again.

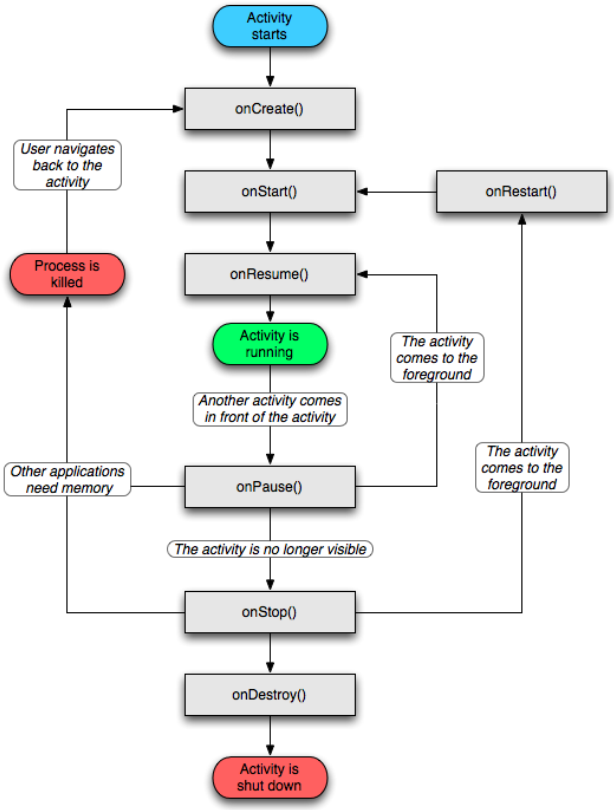

Figure 3: An Android Activity consists of various callback methods that are called if the view or page is created, displayed, sent to the background or destroyed. Source: `http://developer.android.com/`

However, this approach would still require developers to add source code manually thinking in terms of object-oriented programming like it is done for existing frameworks. As a solution to this problem, we leave object-oriented programming and introduce aspect-oriented programming into our mobile framework [7]. This way, framework source code is injected into applications directly before they are compiled using an aspect-oriented compiler (see figure 4). As Android is based on Java, AspectJ, a Java-based aspect-oriented framework, can be used and works well for our purpose [23]. A more technical explanation on how AOP is used to sense user interaction in mobile applications can also be found in [8].

Therefore, we believe that some of the metrics specified for websites can also be considered suitable for Android applications, which is, for instance, demonstrated by Google Analytics[11]. We decided to go for standards and definitions which facilitate analysing and measuring the success of web sites provided by organizations like

[11]http://www.google.com/intl/de/analytics/

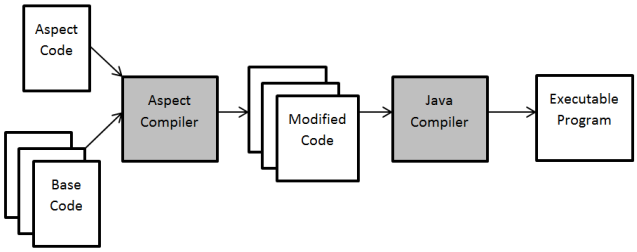

Figure 4: Aspect code is merged with application code at compile time, which is referred to as compile-time weaving. The aspect weaver produces code that can be compiled by the Java compiler.

JICWEBS (The Joint Industry Committee for Web Standards in the UK and Ireland[12]), ABCe (Audit Bureau of Circulations electronic, UK and Europe[13]) and the WAA (Web Analytics Association, US[14]) as well as research work published by Tullis et al. [20].

4.1.1 Metric Categories

In 2007, the WAA published a standard that defines types and metrics for website analysis. Most of the aspects presented by this standard can also be found in [20]. According to this standard, low-level metrics can be divided into two major categories:

- **Counts.** Counts are the very basic unit providing measures. Counts are single numbers that do not express ratios or comparative results (e.g. number of visits, revenues).
- **Ratios.** Ratios involve two different count metrics, where one count is divided by the other. Alternatively, a ratio can also be divided by a count or vice versa (e.g. page views per visit, hit/miss ratio).

4.1.2 Metric Universes

As low-level metrics try to express measures for each and every purpose, different metrics are applied for different scenarios. This is necessary to provide a coarse overview of a system status on the one hand, and to allow interpretations and implications at a finer granularity (e.g. for individual users) on the other hand. Therefore, the WAA distinguishes between three different universes of metrics:

- **Aggregated.** This takes into account the total data traffic of a whole system or website. Aggregated metrics provide a more general overview of an entire system and do not allow detailed assumptions about the behaviour of single users.
- **Segmented.** Segmented metrics are used in order to get a better analytical insight into the system. Segmented metrics collect data from one part of the system and only for a certain period of time. This data is then filtered (e.g. user groups, functional modules, etc.)
- **Individual.** Individual metrics are used to monitor the behaviour of single users for a certain period of time. Furthermore, individual metrics can be seen as a special case of segmented metrics.

[12]http://www.jicwebs.org/
[13]http://www.abc.org.uk/
[14]http://www.webanalyticsassociation.org/

4.1.3 Useful Metrics for Mobile Applications

In order to get comparable data for Android applications, we decided to collect data at the granularity of a page or activity to get results on a user's navigational behaviour. Therefore, low-level metrics are introduced to identify navigational errors or inefficient navigation concepts for existing applications. Compared to different approaches from the area of usability research, our system is designed to generalize well on unknown applications. In other words, our solution should work well on user interfaces without the need for the definition of application-specific tasks, which would be the case with cognitive walkthroughs for example [18].

First, we calculate count-based metrics such as session times, screen calls, button clicks, etc. These metrics can be aggregated as well as segmented for specific user groups or single application users and can therefore be filtered. Additionally, we keep track of device statistics (i.e. application version, operating system version, hardware information, etc.) which enables us – in combination with count-based metrics – to apply filters to compare data from different user groups (e.g. users with different hardware, users with different versions). As single screens can contain panels and other view containers, the metrics are also collected for the entire view hierarchy. Thus, screens are represented as k-ary view trees, where the root node of the tree represents the outer screen and its children represent view containers like panels respectively *Fragments* in Android. This hierarchy is visualised in hierarchical tree maps.

Second, we use these low-level metrics in combination with device statistics to derive usability-related metrics. For example, we use hit rate and miss rate to built the *hit/miss ratio* where we define the miss rate as the total number of screen calls immediately leaving the screen again, where "*immediately*" refers to a defined threshold that can be changed at any time on our server-side user interface by the application developers. The hit/miss ratio can be used for aggregated application statistic, for segmented user groups or for individual users. It is calculated using the total number of hits divided by the sum of misses and hits together. The hit/miss ratio can be seen as a screen-based metric. In terms of this paper, low hit/miss ratios indicate that a screen does not contain relevant information, and might be dismissed or reworked at some point in the design process.

Third, we associate the voluntarily provided user data, which is mapped in user profiles, with recorded metrics. Thus, we can compute usability metrics regarding user satisfaction and loyalty. To be more specific, the information allows us to calculate a customer lifetime value (CLV) [3]. The CLV, for example, sheds light on the user loyalty by showing the difference between new users and returning users per day. Additionally, it allows us to visualize how many users use the application more than once a day.

4.2 Usability Metrics to Identify Navigational Errors

Compared to the frameworks presented in Section 2, our toolkit uses voluntarily provided user data for filters and data recombination in order to create new usability-related visualisations. This allows us to create more complex widgets such as navigation graphs, tree maps or heat maps giving a deeper insight into usability metrics. Moreover, the filter system in combination with voluntary user data allows us to directly compare data from different user groups as mentioned in the requirements section.

According to Tullis et al. [20], usability metrics can be classified as performance metrics. Performance metrics comprise user success rates, time on tasks, errors, efficiency rates and learnability. At first, we want to identify errors in the interface structure of mobile applications using low-level metrics (e.g. importance of single

screens, unused and misleading connections between screens, unused buttons, etc.). Therefore, our current research focuses on the measurement of usability errors without using supervised testing and predefined tasks. Within a mobile application, the most important thing is to be aware of possible navigation graphs and errors that are made while using the application.

In terms of usability, the error rate reflects errors made by users during application usage, but also how these errors influence the work flow. Both Jakob Nielsen and Ben Shneiderman point out that it is important to help users recognize and recover from errors [13, 19].

4.2.1 *Navigation Graphs*

Based on the information we receive from the Android lifecycle, the proposed framework is able to construct a navigation graph (see Figure 5) that can be used to graphically describe either aggregated data (i.e. all graphs submitted over all users), segmented data (i.e. all graphs from a single user) or individual data (i.e. a single graph from a single user). Within our navigational graph, application screens are represented with nodes, and transition between two different screens with edges between the nodes. Thus, we get a directed graph that provides us with information about how users navigate through the system.

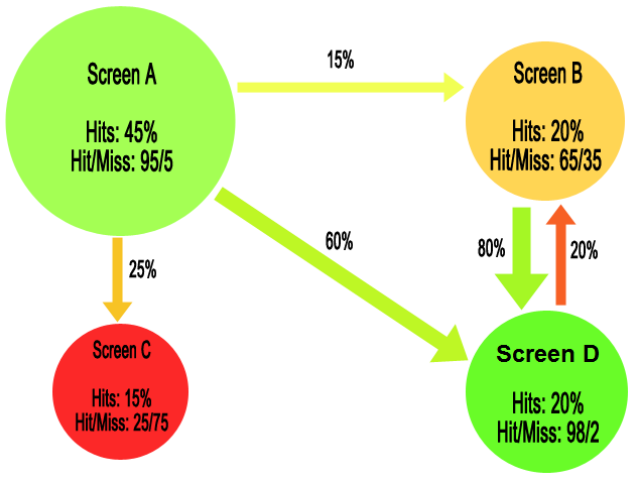

Figure 5: Visualization of an augmented application graph for a demo application. The nodes represent single screens whereas the edges define the possible and also used transitions between the different screens.

4.2.2 *Augmenting Navigational Graphs*

By introducing a consistent system of visually augmenting nodes and edges, semantic meaning can be added to the graph which allows the extraction of navigation errors or possible design flaws at screen granularity.

In order to connect the navigational graphs to semantic information, low-level metrics are used. The hit/miss ratio, for instance, helps identifying less useful nodes and can indicate problems with certain navigation paths that can be caused by misleading localization or other design flaws.

For augmentation, color and size indicators are used to represent hit rates and hit/miss ratios. The sizes of the nodes are interpolated between a minimum and maximum size, so smaller screens indicate less frequently visited screens. Thus, smaller nodes might refer to unnecessary or hardly used screens. Moreover, a traffic light system is introduced. The edges and nodes are coloured according to their hit miss ratio. A red node, for example, indicates that the miss count for a screen is higher than its hit count, which might also refer to unused components. If a miss results in returning to its origin, a so-called *bounce* is detected [21]. Bounces may indicate navigational errors based on too small buttons or inaccurate button labels.

Colour is an extremely effective method for creating emphasis and providing feedback. According to Horton [5], these are two essential aspects of a user interface. The solution is to provide redundant emphasis and feedback using other methods whose meanings are recognized intuitively without the need of further explanation. Based on the example of traffic lights or mechanic gauges, for instance, developers are supposed to associate *red* with negative aspects or problems whereas *green* is more likely to be associated with positive aspects. Moreover, the size of objects is also a valid way of highlighting important objects. According to Lidwell et al. [9], size can be used to draw attention to certain objects.

In addition, colour can also be used in order to produce heat maps for single nodes. Basically, our framework is capable of identifying clicks on a device's touch screen and associate these clicks with components such as buttons or text fields. First, this gives us the opportunity to see which of the components are important and often accessed by users. Second, a heat map that takes the average size of a human finger into account can provide valuable conclusions on the on-page placement of components. It can indicate if users unintentionally interact with components that they did not want to interact with. Thus, we are able to identify common design flaws like too small buttons or too small distances between components which is strongly related to usability errors [20]. Third, by tracking interactions on the touch screen, we are also able to record gestures like scrolling or flinging down a page. By taking this into account, we are able to get information whether users find the information they seek for on top of the activity or whether useful information is located at the bottom of a screen. This information can be used in order to enhance the efficiency of an application as it is very easy to reorganize a screen by relocating its components.

4.3 Discussion

Although low-level metrics can only provide limited relevant data, they seem to be very efficient to derive usability metrics and to draw conclusions on users navigational behaviour and errors they make. The hit rate, for instance, does not necessarily tell anything about navigational problems on its own, but in combination with other low-level metrics such as the miss rate, it can be used to augment the users' navigation graph in order to indicate errors in the navigational concept. Additionally, empirically evaluated thresholds provide the options to individually define for each application how strict the error measurement has to be performed.

Moreover, the missing possibility to define tasks like in scenarios turns out to be a problem when it comes down to the evaluation of other usability metrics such as effectiveness. Normally, for effectiveness the success rate or the lostness is computed within supervised experiments involving the supervisor to count mistakes that occur while a test participant performs a certain test [18]. However, our framework can also be used within supervised experiments, where we do not have to take care of these issues. Moreover, although it seems to be difficult, we are able to take success rates into account by applying a slight modification. Instead of using success rates to describe the overall success rate of an application, competitive success rates can be used to compare the effectiveness of a certain version of an interface to another [10]. As a result, we are able to measure the improvement from one application version to

another one. Additionally, we are currently developing a concept which allows us to define tasks within the proposed framework, so it will be possible to generalize the calculation of other usability metrics than error rates as well. Instead of defining events manually, our idea is based on a record-and-replay system that allows developers to record different tasks and scenarios and compare the input of users against these pre-recorded scenarios.

5. FEASIBILITY STUDY OF THE PRESENTED APPROACH

In order to get results about applicability and reliability, the presented approach was tested against various aspects. First, the evaluation should pull off a successful proof of concept, to demonstrate the framework's feasibility in general. Therefore, we wanted to show that the integration of the proposed framework into a real application does not influence users and can be used for automatic collection of low-level metrics. These metrics should be used to draw conclusions on the navigational behaviour of the tested application (i.e. identify more and less often used components and navigation errors).

Second, we present results from a field study designed to show that our concept of automatic usability evaluation also works in practice. The focus on the field study was on the applicability of low-level metrics and on first usability results regarding navigational errors. Moreover, the feasibility study should highlight difficulties or potential inconsistencies using metrics for usability evaluation. For the feasibility study, an Android application has been developed with our framework. The study was carried out in August 2011, where usability data has been collected for a whole week resulting in 355 application calls.

5.1 The Studied Application

The application which has been used for the study was ScotDroid[15], which is an Android ticketing application for buying train tickets. ScotDroid consists of seven screens (activities) for ordering, administrating and editing tickets. ScotDroid is based on Android 2.3.3 with downward compatibility to Android 2.0, and it was available for free in the Android market on October 11th, 2011, when the study has been carried out.

By adding the presented framework to the ScotDroid application, code is automatically inserted for each of the seven activities. This code is responsible for recognizing when a new screen is created or when an existing screen is destroyed. In this manner, the framework detects whenever the ScotDroid application is launched or terminated by the user. In addition, the framework detects which screen is shown to the user at a certain point in time, how long the user stays on this screen, and when the user moves to a different screen. The framework is aware of each activity and knows which activity is in the foreground or has the focus. As the presented framework detects when the user closes the ScotDroid application, it can send the recorded data to the webserver at that time, or save it temporarily on the phone's internal storage if no network connectivity is available.

5.2 Analysis and Findings

Based on the data transmitted by the ScotDroid application users, usability diagrams were generated. Some minor design flaws could be identified, like for example that users had to wait for about 30 seconds for a ticket due to server delays, without a feedback by the application. These delays could only be identified in the field, and led to the introduction of a status panel which informs the users about the current status.

In addition, we were able to identify frequently used screens and navigation paths, which allowed us to draw conclusions on the average use of the analysed application.

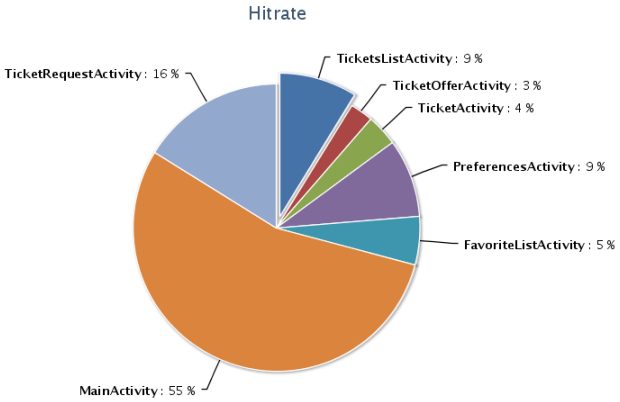

Figure 6: Hit rate for the ScotDroid Android application.

The analysis also showed some very typical characteristics regarding default mobile application architectures. For example, the main menu of an application will always result in a higher hit rate than other screens, which is due to the fact that it is called by default and represents the entry point for the application (see figure 6). However, this does not negatively influence the evaluation process at all, as the root screen of an interface (i.e. the main menu) can be highlighted differently in the visualization to avoid confusions and to clearly identify the entry point of an application.

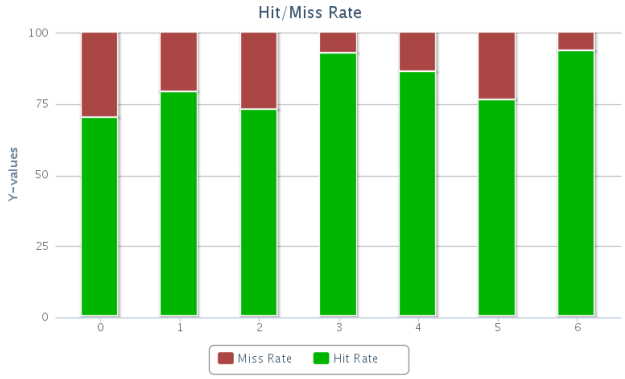

Figure 7: The aggregated hit/miss ratio for the ScotDroid application, showing a bad hit miss ratio for the main activity at position 0.

Additionally, as activities do not necessarily have to be destroyed in Android, a higher miss rate is also achieved for pages that will stay in the background when a different activity is launched. Regarding the analysed application, the miss rate for the main activity is slightly higher, because when users want to close an application from a different activity, they have to go back to the main activity first (see figure 7). As the view concept for Android behaves like a stack, the main activity is closed immediately. This may lead to misinterpretations as in this case going back for closing an applica-

[15]https://market.android.com/details?id=at.fhooe.scotdroid

tion is typically not a page miss. However, this also might not be a big problem, because not the absolute number of hits and misses is relevant but the ratio of both.

To sum up, using low-level metrics for the augmentation of application graphs to draw conclusions on navigation errors, for instance, turns out to be a suitable approach to measure one important aspect of usability. Thus, we believe that it is also possible to apply this strategy to different usability metrics such as learnability, memorability and effectiveness using application scenarios and tasks, which is currently in progress and part of our future work.

6. CONCLUSIONS AND FUTURE WORK

The results from the feasibility study have shown that our framework is a valid approach for the automatic collection of usability data. However, during the feasibility study, we found out that context-awareness on mobile phones is an important aspect which we have not taken into account so far. In particular, Bevan et al. [2] point out that context may influence or even change the usability of a product. They furthermore mention that only some, and not necessarily all, usability attributes are context-dependent. Hence, we can provide usability evaluation for generalized conditions anyhow, treating the context as a separate independent aspect within our research work. Therefore, we are currently extending our framework to detect user context like walking, standing or sitting with built-in sensors from the phone.

Low-level metrics turned out to be a useful way for getting feedback on usability, especially regarding navigation errors, and the users' general navigational behaviour. Thus, it can be assumed that self-explanatory usability metrics can be created by the combination of multiple low-level metrics. However, we suffer from the limitation of not having pre-defined tasks. Therefore, we are currently working on a solution of defining and adding user tasks during the design process. If an application can be compared based on tasks, standard usability techniques such as success rates, lostness, learnability and memorability can be computed automatically as well.

In addition we plan to extend the toolkit to collect more user-specific and voluntary information. For example, we are planning to provide possibilities to compare data from healthy users with data from users with disabilities (e.g. motor-driven diseases) and to compare usability data from left-handed to right-handed users. Therefore, we are enhancing the toolkit by a statistics module to automatically validate the collected data. This statistics module should assist developers in applying suitable filter sets. Statistical tests will show which filters can be applied and which results can be compared in order to get relevant and significant results.

Moreover, we are working on a concept and a first prototype for comparing usability metrics with each other. Therefore, we will evaluate how we can compare different metrics by either merging them in one single chart (e.g. area or line charts) or computing and visualising differences for other types of charts. As a first step, we implement a split view, which allows us to directly compare data based on different filter settings.

Acknowledgments

The research presented is conducted within the Austrian project "AIR – Advanced Interface Research" funded by the Austrian Research Promotion Agency (FFG), the ZIT Center for Innovation and Technology and the province of Salzburg under contract number 825345.

7. REFERENCES

[1] Balagtas-Fernandez, F., and Hussmann, H. A methodology and framework to simplify usability analysis of mobile applications. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering* (2009).

[2] Bevan, N., and Macleod, M. Usability measurement in context. *Behaviour and Information Technology 13* (1994), 132–145.

[3] Gupta, S., Hanssens, D., Hardie, B., Kahn, W., Kumar, V., Lin, N., and Sriram, N. R. S. Modeling customer lifetime value. *Journal of Service Research 9* (2006), 139–155.

[4] Hertzum, M. User testing in industry: A case study of laboratory, workshop, and field tests. In *Proceedings of the 5th ERCIM Workshop on "User Interfaces for All"* (1999).

[5] Horton, S. *Access by Design: A Guide to Universal Usability for Web Designers*. New Riders Publishing, 2005.

[6] Kaikkonen, A., Kallio, T., Kekäläinen, A., Kankainen, A., and Cankar, M. Usability testing of mobile applications: A comparison between laboratory and field testing. *Journal of Usability Studies 1* (2005), 4–16.

[7] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., and Irwin, J. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)* (1997).

[8] Lettner, F., and Holzmann, C. Sensing mobile phone interaction in the field. In *Proceedings of the 4th International Workshop on Sensor Networks and Ambient Intelligence (SeNAml 2012)* (2012).

[9] Lidwell, W., Holden, K., and Butler, J. *Universal Principles of Design: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design*. Rockport Publishers, 2010.

[10] MacKenzie, S., and Read, J. C. Using paper mockups for evaluating soft keyboard layouts. In *Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research* (2007).

[11] Madrigal, D., and McClain, B. Usability for mobile devices, September 2010. http://www.uxmatters.com/mt/archives/2010/09/usability-for-mobile-devices.php.

[12] Nael, M. Design issues for usability of residential multifunction terminals. *IEEE Journal on Selected Areas in Communications 9* (1991), 518 –523.

[13] Nielsen, J. *Usability Engineering*. Morgan Kaufmann Publishers Inc., 1993.

[14] Nielsen, J. Usability metrics: Tracking interface improvements. *IEEE Software 13* (1996), 12–13.

[15] Nielsen, J. Usability 101: Introduction to usability, August 2003.

[16] Nielsen, J. Medical usability: How to kill patients through bad design, April 2005.

[17] Oztoprak, A., and Erbug, C. Field versus laboratory usability testing: A first comparison. Tech. rep., Department of Industrial Design - Middle East Technical University, Faculty of Architecture, 2008.

[18] Polson, P. G., Lewis, C., Rieman, J., and Wharton, C. Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies 36* (1992), 741–773.

[19] Shneiderman, B. *Designing the User Interface: Strategies*

for Effective Human-Computer Interaction. Addison-Wesley Longman Publishing Co., Inc., 1986.

[20] Tullis, T., and Albert, W. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann Publishers Inc., 2008.

[21] WebAnalyticsAssociation. Webanalyticsdefinitions vol1. Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 United States License, August 2007. http://www.webanalyticsassociation.org/.

[22] Wilson, P. I. Active story: A low fidelity prototyping and distributed usability testing tool for agile teams. Master's thesis, University of Calgary - Department of Computer Science, 2008.

[23] Xerox. The AspectJ programming guide. Palo Alto Research Center, 2003.

[24] Zduniak, M. Automated gui testing of mobile java applications. Master's thesis, Poznan University of Technology Faculty of Computer Science and Management, 2007.