Automated Usability Testing for Mobile Applications

Wolfgang Kluth, Karl-Heinz Krempels and Christian Samsel
Information Systems & Databases, RWTH Aachen University, Aachen, Germany

Keywords: Usability Testing, Usability Evaluation, HCI, Automated Testing, Mobile.

Abstract: In this paper we discuss the design and implementation of an automated usability evaluation method for iOS applications. In contrast to common usability testing methods, it is not explicitly necessary to involve an expert or subjects. These circumstances reduce costs, time and personnel expenditures. Professionals are replaced by the automation tool while test participants are exchanged with consumers of the launched application. Interactions of users are captured via a fully automated capturing framework which creates a record of user interactions for each session and sends them to a central server. A usability problem is defined as a sequence of interactions and pattern recognition specified by interaction design patterns is applied to find these problems. Nevertheless, it falls back to the user input for accurate results. Similar to the problem, the solution of the problem is based on the HCI design pattern. An evaluation shows the functionality of our approach compared to a traditional usability evaluation method.

1 INTRODUCTION

In a time when more and more consumers use technical devices to manage their everyday life, usability in software is important. A user friendly handling of smart phones, personal computers and smart televisions depends on the interface between human and computer. The large number of mobile applications in the consumer market leads to increased efforts to improve the usability for mobile devices. Additionally, better and faster hardware leaves mobile devices more capabilities for realizing complex software, but the device and display is still of small size. Thus, it is a challenge to develop appropriate software with good user experience.

In human-computer interaction (HCI) one goal is measuring and improving the usability of soft- and hardware. Different usability testing methods have been developed to estimate the quality of user interfaces and to derive solutions for usability improvements. While evaluations with users (e.g., cognitive walkthrough and heuristic evaluation) and without users (e.g., user observation and think aloud) are widely used, automated usability testing (AUT) is still an untouched area, especially in the method of mobile devices. With and without users, usability testing requires a lot of development time, money and HCI experts. These are reasons and excuses to avoid the integration in software development processes. One solution could be the automation of usability tests with the objective of reducing the efforts of software developers to make usability evaluation more attractive.

The focus of this work is on mobile applications, which, in comparison to personal computers, have additional usability problems due to their mobile context (i.e., in which situation the device is used) (Schmidt, 2000), size and computing power. Nevertheless, it is also a challenge to find appropriate usability testing methods to evaluate mobile applications (Zhang and Adipat, 2005). The development of the mobile device and app market shows the importance of an automated usability evaluation tool.

Our main objective for this paper is the development of a fully automated tool for testing usability problems of mobile applications in the post-launch phase implemented for Apple's[1] iOS platform. It should replace the current evaluation technique *think aloud* which is typical for smart phones.

This paper is structured in six chapters. Section 2 gives an overview of current approaches in AUT. Section 3 and Section 4 describe the theoretical idea and implementation. At the end, in Section 5 the implementation is tested with a bike sharing application prototype and Section 6 reviews this work with an outlook to future work.

[1] http://www.apple.com

Kluth W., Krempels K. and Samsel C..
Automated Usability Testing for Mobile Applications.
DOI: 10.5220/0004985101490156
In *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (WEBIST-2014), pages 149-156
ISBN: 978-989-758-024-6

2 RELATED WORK

In (Ivory and Hearst, 2001) AUT is separated in five method classes: *testing*, *inspection*, *questionnaire*, *analytic models*, and *simulation*. The approach described in this paper concentrates on the class of testing with real users which are involved to evaluate the mobile application. However, we limit the amount of related work in this section to mobile platforms.

According to (Ivory and Hearst, 2001), AUT has four different steps of automation: *nonautomatic*, *automatic capture*, *automatic analysis*, and *automatic critic*. Each automation step is consecutive to its predecessor. Furthermore, the effort of AUT is estimated formally (explicit tasks for participants) and informally (participants use target system without any further tasks).

2.1 Capturing

In (Lettner and Holzmann, 2012a) and (Lettner and Holzmann, 2012b), Lettner and Holzmann present their *Evaluation Framework* for capturing user interactions for the Android[2] environment. The approach works with aspect-oriented programming (AOP) which adds the capture functionality automated within the Android application. With AOP the compiler includes the important logger methods directly into the application's lifecycle.

Another capturing approach is presented in (Weiss and Zduniak, 2007) where a capture and replay framework for Java2MicroEdition (J2ME) environment has been developed. A proxy was used in combination with *code injection* (modify event methods) to intercept graphical user interface (GUI) events. The tool creates a log file, which can be read, modified by the developer, and replayed on a simulator.

2.2 Analysis

Many approaches for websites and desktop computers exist which automatically capture the interactions of the users to analyze them (Ivory and Hearst, 2001). Nevertheless, their functionality is reduced to the visualization of interaction processes and statistical analyses of e.g., resting time in views, number of clicks, and hyperlink selections.

One of the rare AUT tools for analysis of mobile applications is *EvaHelper* (Balagtas-Fernandez and Hussmann, 2009). It is implemented in a four phase model: *preparation*, *collect*, *extraction*, and *analysis*. *Preparation* and *collect* are phases of the automated capturing process and in contrast to approaches in Section 2.1 it needs a manual implementation of the logger methods. Nonetheless, in *extraction* the log is converted into GraphML[3], a machine readable format. This format makes it possible for the developer to apply explicit queries on this graph for analysis.

2.3 Critic

In (Albraheem and Alnuem, 2012) a survey of AUT approaches shows that only one approach exists which implements *automatic critic* for mobile applications. With *HUI Analyzer* (Au et al., 2008)(Baker and Au, 2008) Au and Baker developed a framework and an implementation for capturing and analyzing user interactions for applications of the Microsoft .NET Compact Framework 2.0 environment. *Automatic critic* is implemented in this project in an automatic review of static GUI elements on the basis of guidelines. However, with *HUI Analyzer* it is possible to compare actual interaction data (e.g., clicks, text input, and list selections) with expected interaction data (i.e., series of interactions predefined by the evaluator). With this piece of information the evaluator can check if the user successfully finished a task or failed.

In the field of commercial AUT tools, remoteresear.ch[4] gives a good overview of existing products. Two examples for capture and replay are Morae[5] and Silverback2.0[6]. Furthermore, Localytics[7], Heatmaps[8], and Google Analytics[9] support capture and automated analysis functionality. They include statistics and heatmap visualizations to represent the collection of interaction data. A disadvantage of all tools is the manual integration of capture methods by the developer. None of them supports *automatic critic*.

3 APPROACH

Section 2 gives a good overview of existing approaches and makes clear that no approach exists which uses user interaction data to automatically analyze and critique the usability of mobile applications. In this context, one of the major objectives in this paper is an approach which fulfills this requirement. According to (Baharuddin et al., 2013), the

[2]http://www.android.com

[3]http://graphml.graphdrawing.org/

[4]http://remoteresear.ch/tools

[5]http://www.techsmith.com/morae.html

[6]http://silverbackapp.com

[7]http://localytics.com

[8]http://heatmaps.io

[9]http://www.google.com/analytics/mobile

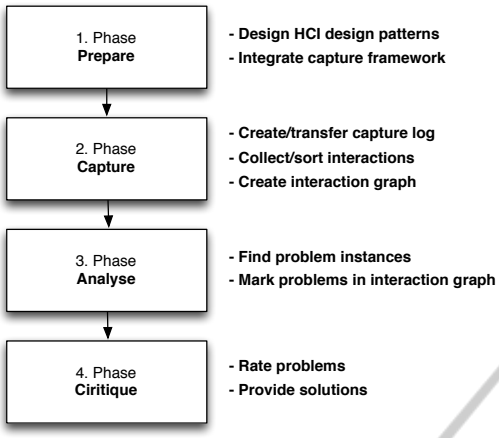

Figure 1: From (Balagtas-Fernandez and Hussmann, 2009) adapted four-phase model for fully AUT tool.

most common approach to evaluate mobile applications is *think aloud* (the user is observed while he is using the application; he talks about what he is actually doing and thinking). What is the method of *think aloud* and how can machines be used to simulate this process to generate similar results? The important steps of this evaluation method is to recognize misuse, usability problems, different and non predicted behavior. The findings are always different because of where they occur. Nevertheless, most of the problems have a significant pattern. Our goal is to determine which problem patterns exist and how they can be found automatically. For this purpose, HCI design patterns (best practices to solve a recurring usability problem)(Borchers, 2000) help to define interaction problems in a usual matter to reuse them for automated analysis and critic. The four-phase model from (Balagtas-Fernandez and Hussmann, 2009) has been adapted with the difference that the phase of extraction and analysis are taken together and a phase of critique has been inserted. The phase of critique allows the developer to get feedback in form of a suggestion for improvement for analyzed usability problems. The purpose of each phase is explained as follows:

1. **Preparation Phase.** To prepare his project, the developer integrates the AUT capture framework into his application. In addition, he needs HCI design patterns to apply them to the captured interactions. He can reuse patterns from an open platform or he can develop his own patterns.
2. **Capture Phase.** Each mobile device will automatically generate logs with the user's interaction data. When the session ends, the application will send the log to a central server where it is processed.
3. **Analysis Phase.** Pattern recognition definition, built from the problem specifications of the HCI design patterns, are applied to each new transmitted log to find and mark related problems.
4. **Critique Phase.** With the problem HCI design pattern relation a solution in form of best practices for each problem is given. With a higher detail degree of problem specification, the precision of the solution increases.

In this paper a lightweight HCI design pattern definition is used. A pattern consists of a name, a weighting, a problem specification, and a solution. According to the Usability Problem Taxonomy from (Keenan et al., 1999), all considered problems are within the *task-mapping* category consiting of interaction, navigation, and functionality. We assume that especially this kind of usability problems can be found and improved with our approach. The solution to the usability problem, an improvement of the usability, is given in form of best practices based on the HCI design pattern. The weighting scale is based on Nielsen's *severity rating* (Nielsen, 1995).

Hence, we have designed four HCI design patterns presented in Table 1 which satisfy the declared attributes.

The capture component has the purpose of collecting user interactions directly on the mobile device of the user (see Figure 4). An interaction consists of seven attributes: startview, endview, called method, user input-type, timestamp, touch position, and activated UI-element. A log (group of user interactions) is transferred to a server when the user session is over (i.e., when the application is closed). The procedure of capturing is automated and similar to (Lettner and Holzmann, 2012b) and (Weiss and Zduniak, 2007).

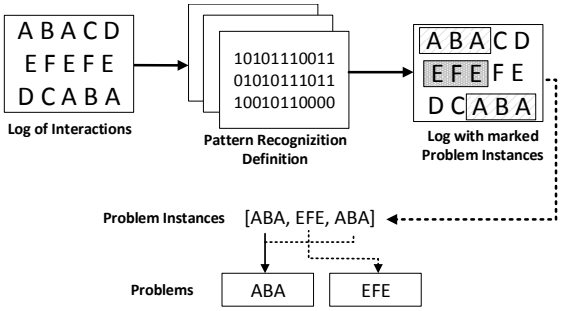

Figure 2: Analyze process with pattern recognition.

In our perspective a usability problem is a sequence of specific user interactions which is defined in the problem specification of a HCI design pattern. E.g., for the pattern *Fitts's Law* all sequences are observed where the user touches repeatedly points next to a button until the button itself is pressed. For this

Table 1: Four HCI design patterns developed for AUT tool.

Name	Problem Specification	Solution	Weighting	Reference
Fitts's Law	User misses UI-Element (e.g., button) several times	Make UI-Element bigger and/or move to center	Major usability problem (3)	(Fitts, 1992) & (Henze and Boll, 2011)
Silent Misentry	User repeatedly touches UI-elements without functionality (e.g., imageview)	Analyze pressed UI-element and figure out which functionality the user intended; e.g., imageview: image-zoom; add functionality or make clear function is missing	Cosmetic problem only (1)	-
Navigational Burden	User switches back and forth between two views multiple times (e.g., master-/detailview)	User is looking for some information which is presented in detailview; needs the way over the masterview to open a new detailview	Minor usability problem (2)	(Ahmad et al., 2006)
Accidental Touch	User touches the screen accidentally and and activates view change; he immediately revokes input	Check accidentally touched UI-element; move/resize/remove it	Cosmetic problem only (1)	(Matero and Colley, 2012)

purpose, we use pattern recognition with the user interaction log as input word and the sequence we are looking for as search word (s. Figure 2). Furthermore, it is important to mention that the recognizer looks for dynamic keywords, because in the last example, the views, and the button can change, but it is still a sequence of the same problem and just another instance. As a consequence, we defined an advanced regular expression with dynamic behavior which generalizes the sequences for each pattern. To recognize the usability problems in the pool of interactions, we built from the problem specification of each HCI design pattern a pattern recognition definition based on a regular expression (RE):

- **Fitts's Law:**
  $(\mathtt{A}^{+}\mathtt{B}); \mathtt{A} := (\mathtt{a},\mathtt{e},\mathtt{a}); \mathtt{B} := (\mathtt{a},\mathtt{x},\mathtt{b})$
- **Navigational Burden:**
  $(\mathtt{AB})^{+}; \mathtt{A} := (\mathtt{a},\mathtt{x},\mathtt{b}); \mathtt{B} := (\mathtt{b},\mathtt{z},\mathtt{a})$
- **Accidental Touch:**
  $(\mathtt{AB}); \mathtt{A} := (\mathtt{a},\mathtt{x},\mathtt{b}); \mathtt{B} := (\mathtt{b},\mathtt{z},\mathtt{a});$
  $\Delta\mathtt{t}(\mathtt{A},\mathtt{B}) < \mathtt{t}_0$
- **Silent Misentry:**
  $(\mathtt{A}^{+}); \mathtt{A} := (\mathtt{a},\mathtt{e},\mathtt{a})$

For the sake of simplicity, we reduced the complexity of the regular expression with a simpler definition of interactions. The interaction A in this case consists of a tuple $(startview \times executedmethod \times endview)$ and $\mathtt{e}$ represents no-action. The RE $(\mathtt{A}^{+}\mathtt{B})$ describes a search pattern which starts at least with one interaction $\mathtt{A}$ and ends with one $\mathtt{B}$. Additional comparisons, e.g., timespan and distance, have to be done manually.

Each problem instance is part of a problem and each problem is derived from a HCI design pattern which has a solution for it. The accuracy of a solution depends on the accuracy of a problem specification.

For a better visual communication with the developer, a finite state-machine has been designed where all interactions are represented. Hence, nodes stand for views and edges for user-input with executed method (methodname) or no method ($\mathtt{e}$). In addition, each problem instance is a sequence of interactions and can be marked in the graph to indicate in which state a problem occurs for a better understanding (s. Figure 3). As a result, the developer knows immediately in which states a usability problem occurs, the kind of usability problem, and how it could be solved.

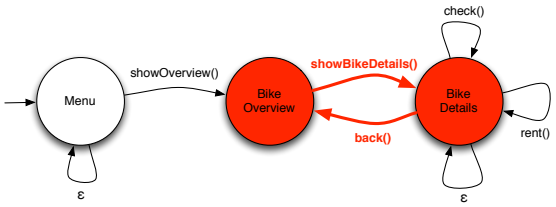

Figure 3: Example for an interaction graph with marked *Navigational Burden* problem.

Figure 4 represents the composition of all phases in one architecture. There are two separated workers, the capturing framework on the client's mobile phone and the central computer unit which evaluates the collected information. Furthermore, the server presents the results of the evaluation in form of a dashboard to the developer.

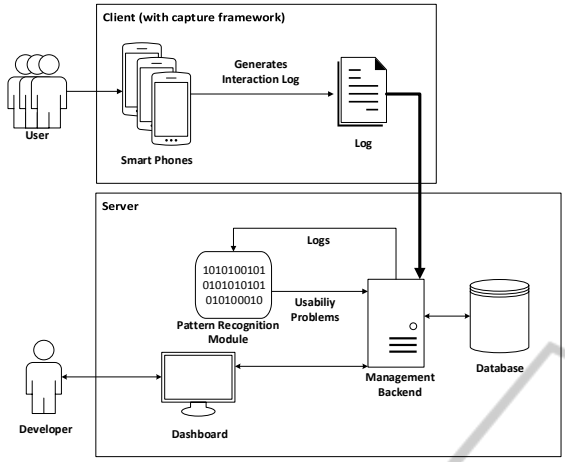

Figure 4: Architecture of the AUT tool.

4 IMPLEMENTATION

Figure 5 shows an overview of all used components for the implementation of the AUT tool. The capture framework has been implemented for Apple's iOS7 environment, uses *method swizzling*[10] and a *gesture recognizer* (Apple, 2013) to identify user interactions.

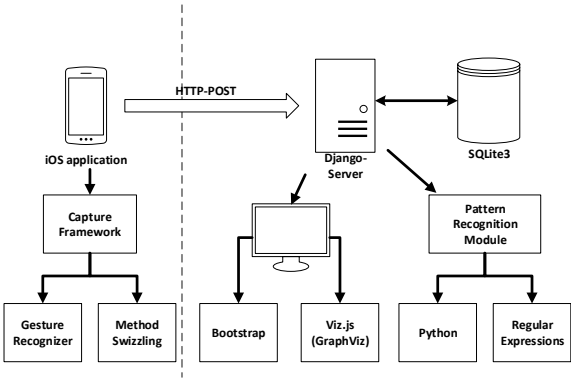

Figure 5: Implementation of the AUT tool.

The log of user interactions is sent via an HTTP-POST to a Python Django 1.5[11] webserver. Each capture log is processed in three steps:

1. Loading the interactions into a database
2. Searching for problem instances with pattern recognition modules in collection of interactions
3. Mapping problem instances into suitable problem categories related to their HCI design pattern

The visualization of the information is dynamically generated from database and presented on a dashboard for the evaluator/developer. The interaction graph is visualized with the help of Viz.js[12] (a GraphViz[13] implementation in JavaScript) and usability problems are marked in red.

4.1 Capture Framework

For the implementation of the capture framework for iOS7 we had to define the interaction attributes and the technique to gather them. Table 2 shows an overview of the type, name, and technique of an interaction attribute. In our context, an advantage of the iOS environment is the Objective-C runtime which works with messages instead of direct method calls when executing a specific method in the application.

Table 2: Interaction Attributes with Type and used Technique.

Type	Name	Technique
NSNumber*	id	GR
BOOL	hasMethod	MS
NSString*	methodName	MS
NSString*	viewControllerTitle	MS
NSString*	viewControllerClass	MS
NSTimeINterval	timestamp	GR
CGPoint	position	GR
NSString*	uiElement	GR

GR = Gesture Recognizer; MS = Method Swizzling

With *method swizzling* the target method is replaced by another method. This procedure allows us to extend methods of private classes which are responsible for the lifecycle of a view controller and the execution of events with capture functionality (s. Table 3). For this purpose, *categories* (Objective-C; extend existing classes with methods) are used to add a new method with capture functionality to the private class and to call the intended method afterwards. On the other hand, a *gesture recognizer* (UIGestureRecognizer) which automatically identifies a user tap (a single touch event) is added to the root view controller of an application (i.e., foreground view in size of the screen). This allows to recognize when a touch begins and ends in all views. In each start/end callback, the touch event (instance of UIEvent) is included. Table 4 shows the attributes which are set in the delegate methods *touchesBegan* and *touchesEnded*.

Single interaction objects are collected in an array which is sent to the backend with a standard HTTP-POST in JSON format when the application is sent to background.

[10] http://cocoadev.com/MethodSwizzling
[11] https://www.djangoproject.com/
[12] https://github.com/mdaines/viz.js
[13] http://www.graphviz.org/

Table 3: Overview of extended private classes.

Category	Method	Attribute
NSObject +Swizzle.h	responds ToSelector	hasMethod
UIApplication +Swizzle.h	sendAction	methodName
UIViewController +Swizzle.h	viewDidAppear	viewController-Title
UIViewController +Swizzle.h	viewDidAppear	viewController-Class

Table 4: Attributes set in Gesture Recognizer Delegate Methods.

Delegate Method	Attribute
touchesBegan	*init new interaction*
touchesBegan	id
touchesEnded	position
touchesEnded	timestamp
touchesEnded	uiElement

4.2 Pattern Recognition Module

Pattern recognition is implemented in Python[14] and has the purpose of finding problem instances in list interactions. Each module implements the method `find_problems_for_pattern(interactions)` which gets a list of interactions as input and returns a list of problem instances. The detailed implementation of the module is provided by the developer. He can use regular expressions, python code or other tools, to describe the target pattern. We designed two example patterns with RE and two programmatically. For the *Fitts's Law* module interactions are converted into string tuples (start-view, executed method, end-view) and the following RE is applied:

```
(\[(?P<view>\w+), no_action, (?P=view)\],)+
(\[(?P=view), \w+:?, \w+\])
```

4.3 Management Backend

The backend manages all incoming data. The backend tasks are maintain database, persist user interactions, integration of pattern recognition modules, and visualize results in an interaction graph.

4.4 Data Visualization

The database stores the interactions of all users. To visualize this information in an interaction graph, the attributes of all interactions and problems are reduced to start-view, executed method, and end-view. This makes it possible to remove doubled interactions. Afterwards, the structure of the graph is generated as a DOT[15] defined string which can be embedded into the dashboard for the developer. On client side, the Viz.js framework, a GraphViz implementation in JavaScript, transforms the DOT definition in a perceptible interaction graph (s. Figure 6). Recognized usability problems are marked as red.

5 EVALUATION

We compare and evaluate the traditional *user observing* evaluation method in contrast to our AUT tool with a bike sharing prototype application. The application is designed to find problems defined in the four HCI design patterns in Section 3. The flowchart in Figure 7 shows the views and connections of the application. The image with the title "Fahrrad-Verleih" (1st view) provokes a *Silent Misentry*, the lent status in the detailview (3rd and 4th view) causes *Navigational Burden* and the small checkbox in the detailview engenders *Fitts's Law*. The application is developed for iOS7 and the capture framework is integrated and initialized. While the AUT tool generates interaction logs automatically, we had to design a questionnaire for the *User Observing* method which takes care of the user's behavior considering the usability problems in the four HCI design patterns. Eight participants aged between 23 and 28 years, two researchers and six computer science students attend the evaluation. All of them has experience with smart phones, half with Android and half with iOS.

The participants got the target to rent a bike with this application. The task specifies a context for this evaluation, but it is not necessary for the AUT tool. In our eyes it is still an informal test in the lab and the user can use the app as a normal bike sharing app. The results in Table 5 show that the AUT tool and the classical *user observing* identify all seeded usability problems. In contrast to an automated capturing solution, we had the feeling that *user observing* makes it difficult to capture all user inputs on a touch screen interface. Getting a clear analysis of the *user observing* data was tedious, because we manually sorted it into the classification of interaction design patterns. However, the AUT tool gave us an overview of all problems for each pattern with an appropriate interaction graph.

[14] http://www.python.org/

[15] http://www.graphviz.org/content/dot-language

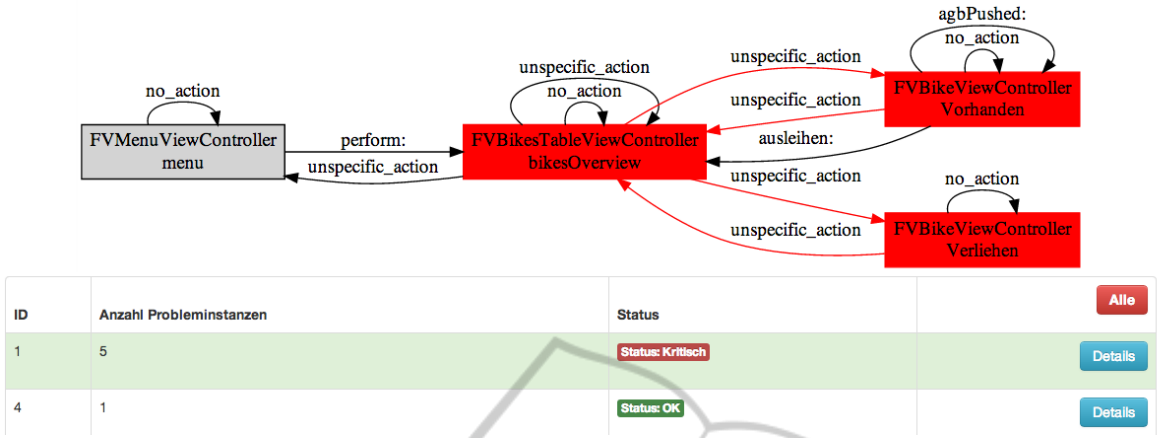

Figure 6: Screenshot of the dashboard visualization for *Navigational Burden*.

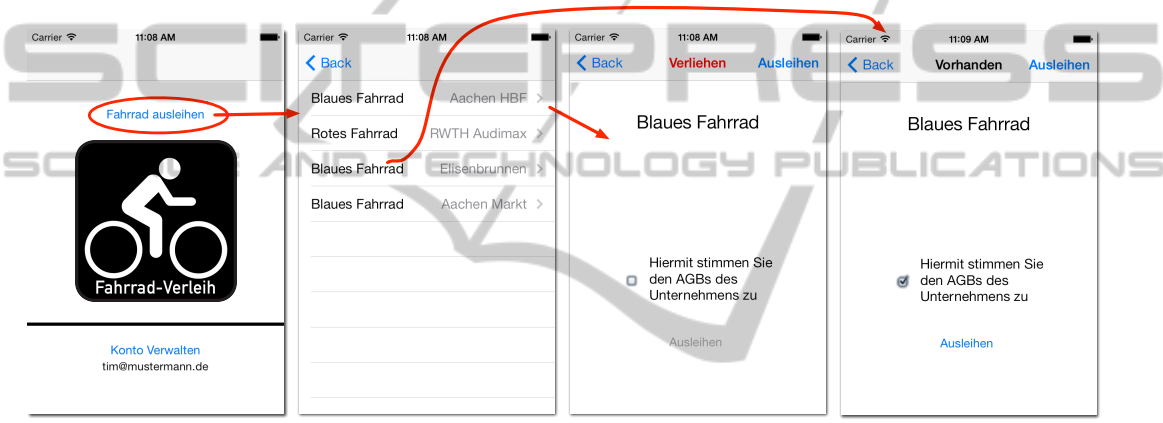

Figure 7: Bike sharing application for iOS7 prepared with usability problems.

Table 5: Results of Bike Sharing App evaluation with *User Observation* and AUT tool.

HCI design pattern	A	B	C
Fitts's Law	2	9	26
Accidental Touch	5	32	-
Silent Misentry	1	4	11
Navigational Burden	2	6	17

A = number of problems (AUT tool)
B = number of problem instances (AUT tool)
C = number of problems (user observing)

6 CONCLUSIONS AND FUTURE WORK

Today, usability is one of the major topics in software development and the relevance for mobile applications is still increasing. Automatic testing, e.g., unit tests, shows that it has a huge influence on the software process and quality. With automated usability testing (AUT) we believe that usability testing will become a firm part of the software development process.

The objective was to develop an automated usability testing tool for iOS applications to get a cheaper, faster and simpler usability evaluation process. We found no existing approaches which fulfill the criteria of automatic critic for mobile applications which takes the responsibility of usability decisions from the developer to a tool.

For this purpose we modified the four phase model from (Balagtas-Fernandez and Hussmann, 2009) and extended it with a phase of *automatic critic*. Usability problems are recognized by pattern recognition which works with definitions from four exemplary HCI design patterns. We implemented a fully automated capture framework for iOS applications and a backend for data management, analysis and representation.

We evaluated the AUT tool using a prototype bike sharing application and compared it to a classic evaluation method, *user observing*. The results show that the AUT tool is able to find all usability problems which are described in the interaction design patterns.

In contrast to *user observing*, working with the AUT tool was less complicated and time-consuming for the developer, which, for us, is an evidence for success.

Future Work

In the next version of the AUT tool, we are going to improve the visualization of the graph by showing a screenshot of the current view as node and the edge will begin directly from the point where the user touched (similar to Figure 7). In addition, we will create more HCI design patterns and a tool which makes the design process much simpler. We also plan to integrate existing fully automated capture frameworks for other mobile platforms (s. Section 2.1).

ACKNOWLEDGEMENTS

We would like to thank our former colleague Paul Heiniz for his support in the early phase of this project. This work was supported by the German Federal Ministry of Economics and Technology[16]: **(Grant 01ME12052 econnect Germany, Grant 01ME12136 Mobility Broker)**.

REFERENCES

Ahmad, R., Li, Z., and Azam, F. (2006). Measuring navigational burden. In *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*, pages 307–314.

Albraheem, L. and Alnuem, M. (2012). Automated Usability Testing : A Literature Review and an Evaluation.

Apple (2013). iOS Developer Library: UIGestureRecognizer Class Reference.

Au, F., Baker, S., Warren, I., and Dobbie, G. (2008). Automated Usability Testing Framework. volume 76, pages 55–64.

Baharuddin, R., Singh, D., and Razali, R. (2013). Usability Dimensions for Mobile Applications-A Review. *Research Journal of Applied Sciences, Engineering and Technology*, 5(6):2225–2231.

Baker, S. and Au, F. (2008). Automated Usability Testing Using HUI Analyzer. pages 579–588.

Balagtas-Fernandez, F. and Hussmann, H. (2009). A Methodology and Framework to Simplify Usability Analysis of Mobile Applications. pages 520–524. IEEE.

Borchers, J. O. (2000). A pattern approach to interaction design. pages 369–378.

[16]Bundesministerium für Wirtschaft und Technologie (BMWi) http://www.bmwi.de/

Fitts, P. M. (1992). The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology: General*, 121(3):262–9.

Henze, N. and Boll, S. (2011). It Does Not Fitts My Data! Analysing Large Amounts of Mobile Touch Data Niels. *INTERACT 2011, Part IV*, pages 564–567.

Ivory, M. and Hearst, M. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys*, 33(4):470–516.

Keenan, S. L., Hartson, H. R., Kafura, D. G., and Schulman, R. S. (1999). The Usability Problem Taxonomy: A Framework for Classification and Analysis. *Empirical Software Engineering*, 4:71–104.

Lettner, F. and Holzmann, C. (2012a). Sensing mobile phone interaction in the field. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 877–882.

Lettner, F. and Holzmann, C. (2012b). Usability evaluation framework: Automated interface analysis for android applications. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory - Volume Part II*, EUROCAST'11, pages 560–567, Berlin, Heidelberg. Springer-Verlag.

Matero, J. and Colley, A. (2012). Identifying unintentional touches on handheld touch screen devices. pages 506–509.

Nielsen, J. (1995). Severity Ratings for Usability Problems.

Schmidt, A. (2000). Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 4(2-3):191–199.

Weiss, D. and Zduniak, M. (2007). Automated integration tests for mobile applications in java 2 micro edition. pages 478–487.

Zhang, D. and Adipat, B. (2005). Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications. *International Journal of Human-Computer*, pages 293–308.