Automated Usability Testing Framework

Fiora T. W. Au, Simon Baker, Ian Warren, Gillian Dobbie

Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand
Private Bag 92019, Auckland, New Zealand

{ian-w,gill}@cs.auckland.ac.nz

Abstract

Handheld device applications with poor usability can reduce the productivity of users and incur costs for businesses, thus usability testing should play a vital role in application development. Conventional usability testing methodologies, such as formal user testing, can be expensive, time consuming and labour intensive; less resource-demanding alternatives can yield unreliable results. Automating aspects of usability testing would improve its efficiency and make it more practical to perform throughout development.

An automated usability testing tool should capture as input the properties of an application's graphical user interface, the sequence of user actions as they use the application to achieve particular tasks, their behaviour and comments, as well as a description of these tasks. The tool should evaluate both the static and dynamic properties of the interface, examine navigational burden and suggest modifications or templates that would improve usability. Results should be quick and easy to interpret, and be understandable by personnel other than specialised testers.

Several existing tools that are typical of the tools available today meet some but not all of these requirements. In this paper we describe the design of the HUIA testing framework, in which we have to meet as many of these requirements as possible.

Keywords: usability testing, tool support, handheld device.

1 Introduction

Handheld devices continue to feature in most companies' mobile business solutions, as a means of improving their workers and thus the companies' productivity. Handheld device applications (HDAs) with poor usability can undermine the value of such solutions, but despite this usability testing is often neglected due to its relatively high demand in time and resources. These issues apply also to functional testing, for which many automated testing tools and frameworks, such as JUnit, were developed and are now widely used making the process far more efficient. Usability testing would also benefit greatly from automation.

This paper examines the need for automating aspects of usability testing. This is followed by discussion on the functionalities that an automated usability testing tool should have, as well as the associated challenges. The design of the HUIA testing framework, a prototype automated usability testing framework for HDA, is then described, and the paper concludes with a brief examination of how this framework as well as other existing testing tools meet the requirements previously discussed.

1.1 Usability

Usability can be defined by IEEE (1990) as '*the ease with which a user can operate, prepare inputs for, and interpret outputs of a system or component*', and comprises of five main attributes as outlined in Le Peuple and Scane (2003) and Nielsen, J. (1993): Learnability, Efficiency, Memorability, Errors, and Satisfaction.

The very nature of handheld devices make their programs particularly susceptible to poor usability; the smaller size, lower resolution screens, limited memory, lack of keyboard and mouse, and frequency of use among other hardware constraints and usage factors make program and user interface design particularly challenging as reported in Weiss (2005), Lee and Grice (2004) and Moe, Dwolatzky, and Olst(2004).

1.2 Usability testing

Many usability testing methodologies exist, with varied opinions on their effectiveness and practicality. Heuristic evaluations are relatively cheap, quick and easy to carry out, but it has been claimed by Le Peuple and Scane (2003), Scholtz (2006), and Spolsky (2001) that such evaluations are likely to identify only approximately fifty percent of actual problems, with a significant number of false problems raised and actual problems missed.

When carried out correctly, user testing is likely to identify most of the major usability issues, as it involves real users attempting real tasks, and is very useful for collecting feedback on subjective aspects of usability (such as satisfaction and aesthetic appeal). Studies by Scholtz (2006), and Spolsky (2001) have shown that six to eight test users (per type of intended users for that particular software) are usually sufficient to identify most of the major usability issues. The type of analyses performed on the gathered data depends on the goal of the user test. Typical areas that are examined include mistake and failure rates, types of mistakes, time taken, amount of interactions (such as number of clicks or

amount of scrolling), user behavior and user feedback as reported in Le Peuple and Scane (2003), Nielsen, J. (1993) and Spolsky (2001). Scholtz (2006) reports that because user tests are expensive and time consuming to conduct, they are usually performed infrequently and towards the end of the development process.

A whole range of quantitative statistics can be extracted or recorded for a given interface; some refer to the properties of the interface, and others refer to user interactions with the interface. Listed below are examples of the two types of statistics:

Interface Properties:

- Number of fonts used, font sizes
- Average size of buttons
- Deepest level of menus
- Average loading time of graphics

Interaction Statistics (for performing specified tasks)**:**

- Number of clicks
- Average drag distance
- Amount of scrolling
- Error rate
- Failure rate

Such statistics however are rarely used on their own as a way of evaluating usability, since numbers have little meaning unless they are placed in context; for example, five clicks may be considered acceptable for accomplishing one task, but too many for another. Metrics therefore are most useful as a rough indication of usability only, and as supporting data for other usability evaluations.

2 Motivation

Handheld devices are playing an increasingly important role in facilitating efficient information exchange, making them a significant driving force in mobilizing businesses and improving productivity as outlined by Lee and Grice (2004) and Moe et al. (2004). However, the value of any application would be undermined if the user is not able to fully utilize its functionality, thus the role of usability testing should be equally important in HDA development as functional correctness.

2.1 Case Studies

We present three case studies that highlight some of the usability issues commonly found in HDAs.

2.1.1 Case study 1: Registration form

Purpose:

Figure 1 shows a simple form on a health and fitness website that users fill in for registration. We would not necessarily expect the users to have high computer skills.

Usability issues:

1. The form requires high precision placement of the stylus, with multiple checkboxes as well as the *Submit* and *Cancel* buttons placed close together, making it easy to press the wrong button or to check the wrong checkbox.
2. Required fields are not marked (*Username*, *Password* and *Confirm password*).
3. The *Password* and *Confirm password* fields are positioned below the *Submit* button, which misleadingly implies that they are not required for the registration process.

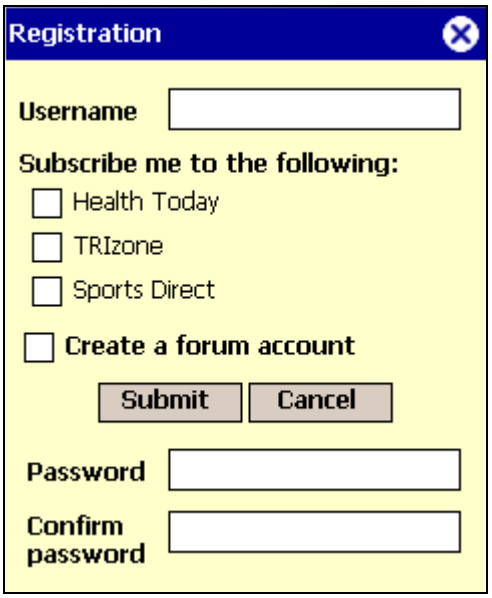

Figure 1: Registration form

2.1.2 Case study 2: Products order placement

Purpose:

The form in Figure 2 allows users to search for products using various filters (search by a combination of product category, brand name or SKU), view information on the products returned, and then to specify the quantity that they would like to order for particular products using the number bar positioned towards the bottom of the form.

Figure 2: Products order placement

Usability issues:

1. Precise placement of the stylus is necessary to specify an amount to order for a product.
2. Columns are difficult to expand to increase the viewable area.
3. The filter droplists are not labelled.
4. There is no clear indication of how to finalize/save an order.
5. The payment method is not marked as a required field.
6. The data displayed in each column, particularly the category and description, are rarely viewable in full. Expanding the columns to increase viewing area is difficult and tedious.
7. Some columns (showing the price, brand) are initially completely collapsed and hidden from view.
8. Items whose width exceeds the width of the droplist are simply truncated, with no indication that they are not viewable in full.
9. Ordered products (ie. for which a quantity to order has been specified) are distinguished from non-ordered products only by the presence of number in the QTY column, which can be easily missed.
10. A quantity of zero is also displayed (if changed to zero from some other original quantity), which at a glance suggests that the product has been ordered, when in fact a quantity of zero means that it has not.

2.1.3 Case study 3: Retailer home page

Purpose:

The display in Figure 3 is the home or start page for a contemporary furniture retailer's online shopping site, providing information for a person who is wanting to find out more about the company and the products they sell. This case study highlights the conflict between form and function apparent in many HDAs.

Figure 3: Retailer home page

Usability issues:

1. High resolution images are not supported by all handheld devices.
2. The blocks of text vary in shape and size, and are distributed all over the page. This reduces the overall coherence of the text and makes it difficult for users to scan the text for information.
3. The text is not very readable due to the small font size and poor contrast between the foreground and background colours.
4. The *Next* >> link requires high level of precision to click, and does not stand out well.

The three case studies present only a subset of the many usability issues associated with HDAs. Clearly, usability testing should feature during product development.

2.2 Automating usability testing

There are many challenges and issues associated with traditional usability testing methodologies, many of which are to do with inefficiency, management complexities and high resource demands; all these contribute to the industry's general reluctance to integrate usability testing as an essential activity on par with functional testing, despite its importance. Instead, it is often considered as a 'nice-to-have' reserved for larger projects with generous budgets.

Conducting usability testing towards the end of the development process runs the risk of leaving insufficient time and resources to respond to the usability issues raised. On the other hand, early tests are often performed on simulators or low fidelity prototypes, which undermine the validity of the tests, and there is a chance of introducing usability defects in later iterations unless regression testing is performed frequently. Agile development promotes this type of iterative testing, for which usability testing would greatly benefit, but this is impractical for most conventional usability testing methodologies according to Kane (2003). For example, user testing is arguably the most effective testing methodology, but it is very inefficient to carry out. The process is labour intensive and costly, having to design the test, recruit suitable test users, set up test sessions, run the test, collect the data and then analyze the results. Other testing methods such as heuristic evaluations and cognitive walkthroughs involving just one or several usability experts are less costly, but not to the point where it can be performed on a frequent, iterative basis.

Currently there is a distinct separation of responsibilities between developers and usability testing specialists. This introduces additional management issues in order to effectively coordinate the activities of the two parties. These specialists may or may not have a good understanding of the application domain, which could affect the validity of their opinions and findings. Furthermore, developers who do not play an active role in the usability evaluations end up learning of their product's usability 'second-hand', having to base design decisions on their own (sometimes incorrect) interpretations of usability evaluation results. This also means that the developer does not gain as much

knowledge as he/she otherwise can, and is less inclined to consider usability to be an integrated part of the development process.

A logical solution would be to automate as many aspects of usability testing as possible, in much the same way that aspects of functional testing has been automated as described in Patton (2006). There has been some excellent work that investigates the state of the art in automated usability evaluation, such as Ivory and Hearst (2001). The challenge is thus to create an automated usability testing tool that is aimed for use by developers, rather than just usability testing specialists, for regression testing through development. The next two sections discuss in further detail the objectives and functional requirements of such a tool.

3 Objectives

The main objectives of an automated usability testing tool are similar in many ways to any automated software tool as outlined in Patton (2006), and may be summarized as follows:

1. **Effective usability testing:** The tool should detect as many if not more real usability issues as conventional usability testing methodologies.
2. **Increased speed:** The tool should perform analyses (checks, calculations, comparisons etc.) and other processes quicker than if they were performed manually.
3. **Increased efficiency:** By taking over parts of the testing process, developers/testers are free to perform other tasks, and need to dedicate less time to testing.
4. **Improved accuracy and precision:** Results produced will always be functionally correct; errors are limited to those produced by inappropriate input or direction by the developer/tester.
5. **Reduced resource demand:** Automation should reduce time, human resources, equipment and cost requirements.
6. **Increased flexibility:** The tool should perform testing on a range of usability aspects, and allow customisation of test settings. It should also facilitate usability testing for all stages of development (as well as iterative/regression testing).
7. **Consistency:** The tool should provide a means of maintaining set testing standards throughout development, so that testing can be performed by different people but the same standard of usability is enforced.
8. **Promote usability:** By simplifying the usability testing process, the tool should encourage all personnel related to product development (including developers, managers, testers, sales personnel and clients) to be involved in the process. Also, the tool itself should promote good usability practices by encouraging the use of proven design paradigms.

4 Functional requirements

The following sections describe the main functionalities that an automated usability tool should offer in order to meet some of the objectives outlined above.

An important architectural decision in implementing the tool would be to adopt a modular design structure; plug-and-play modules for different functionalities would give developers maximum flexibility in customizing usability evaluations for different HDAs. Also, the effectiveness or relevance of each type of evaluation method would differ for different stages of development so the suite of usability tests should be easy to adjust accordingly.

4.1 GUI evaluation

Here, the GUI (Graphical User Interface) of an HDA refers only to the properties and layout of the controls, text and images on a form; the underlying model of the application is not considered. The GUI may be considered as the major determinant of an HDA's readability, so readily affects the HDA's efficiency and subjective appeal.

Some examples of these GUI properties include:

- **Fonts:** Font size, number of fonts
- **Menus:** Number of menu items per menu, menu depth
- **Buttons:** Button size, Number of buttons per form
- **Layout:** Amount of white/empty space on the form, distance of the controls from the edge of the form

These properties should be examined to ensure that they abide to given heuristics, standards or style guides. For example, examining the font size of the case study in Section 2.1.3 would have revealed that it is too small and thus not very readable. Programmatically examining the column widths of the table in the case study in Section 2.1.2 would have drawn attention to the hidden column; a flaw likely to be missed by human inspection.

4.2 Navigational burden

According to Ahmad et al. (2006) a very important aspect of usability to consider is how much effort the user must make to locate and utilize information and functionality, or the navigational burden. Many factors contribute to navigational burden; whilst the GUI also plays a critical part, the underlying model of the HDA, the context of its use and the tasks to be achieved must now also be considered.

4.2.1 Form layout

It was difficult for users to efficiently scan for information in the case study in Section 2.1.3 due to the positioning of the text blocks. And similarly, illogical placement of the input controls in the case study in Section 2.1.1 made it difficult for users to fill in the registration form. The tool should be able to provide some means of identifying the user's glance or interaction sequence (discussed in further detail in Section 4.4.2 *User*

interaction sequences), then together with information of the positions of the corresponding controls on the form, determine if the placement of the controls is appropriate. Kurosu and Kashimura (1995) claim the ideal sequence that is most commonly accepted is top left to bottom right.

The tool should also encourage sensible grouping of controls related to the same task. For example, checkboxes for selecting groceries for purchase should be grouped by food type. This would require input of information about the problem domain and the task to be achieved, which is also further discussed in Section 4.4.4 *Task definitions*.

In the case study in Section 2.1.1, it is easy to press the *Cancel* button by mistake as it is placed too close to the *Submit* button. Kurosu and Kashimura (1995) claim that flagging this and similar error-prone areas would also be useful.

4.2.2 User interaction sequences

Another perspective on navigational burden would be to consider the make-up of the sequence of user interactions necessary to complete a task. Excessive amounts of clicks, scrolling, user input (textual or selection) and forms accessed all indicate poor efficiency; these and other metrics should be examined against predefined tolerance thresholds and be brought to attention if they exceed the threshold.

One of the most important statistics obtained from a conventional user test is the number of errors made, or the ratio between successful interactions and errors. It is arguably the best indicator of usability since it is a reflection of how successful the user was in using the AUT (Application Under Test) to complete his/her task. In a conventional user test this is obtained by an observer manually noting down and categorizing the mistakes that the user makes; many of the issues in the case study in Section 2.1.2 would be made apparent by noting the mistakes that the user makes in attempting to search for and order products. The process however is time-consuming, error-prone and subjective (there are varying definitions of a 'mistake'), making it inefficient and impractical for iterative testing. Thus this area is one that should definitely be addressed by an automated testing tool.

Firstly, the tool should better facilitate the recording of user interactions with the AUT; this is discussed in further detail in Section 4.4.2 *Recording user action sequences*. Secondly, the tool should provide a way for developers to specify without ambiguity what a 'recoverable mistake' and a 'task failure' is. This would require generating a 'correct' or 'expected' interaction sequence against which the test user's interaction sequence can be compared. Related to this is the challenge of designing appropriate test oracles, particularly the grain of comparison. A crude comparator that reports any difference would be least expensive, but likely to produce too many false positives; more sophisticated oracles that intelligently ignore negligible or anticipated differences would be more appropriate, but would be algorithmically complex and difficult to make generic. For example, in the case study in Section 2.1.1 it is not important which optional mail list subscription checkboxes a user checks, but neglecting to check the *Create a forum account* checkbox should be considered either a 'recoverable mistake' or a 'task failure' (depending on whether or not the task is eventually completed) if the task was to register and create a forum account.

4.3 Patterns and templates

The cost and risks of making modifications to the code increase the closer it is to completion. This is also true for resolving usability issues, especially when attempting to make non trivial changes to user interaction sequences that involve new controls or even forms, thus considerations for usability should begin as early as the definition of the problem domain and the tasks that the HDA's users are meant to achieve using it. A way in which the tool could promote good usability would be for it to provide GUI patterns or templates suitable for a given problem domain or task definition as described in Kane (2003). For example, the case studies in Sections 2.1.1 and 2.1.2 are both representative tasks for which HDAs are commonly developed to support. Tried and true GUI and model design solutions for these tasks already exist, analogous to design patterns used to structure program logic and system architecture, and which are widely adopted across the industry. Similar paradigms should be developed for GUI designs and other usability aspects, for example form layouts, font styles, color schemes, page flow guides, link/button maps and menu structures. The tool should present templates or guidelines that describe such solutions, to provide developers with a sound starting point for development and to reduce the amount of code they must write. Using recurring solutions would also promote consistency and thus familiarity for users.

The tool should also provide some way of verifying that the pattern is implemented correctly, and that its form is not broken as the HDA evolves during development. Blewitt et al. (2005) discuss various existing automated verification approaches, including behavioral definition, metaprogramming definition and declarative constraints.

4.4 Tool input and data collection

In order to evaluate the usability of an application or to promote good usability given a particular problem or scenario, the tool needs to collect then interpret information about various aspects of the application. This information ranges from specific to general; from details of the HDA's implementation at control level, to user interaction sequences with the forms, to high level descriptions of the problem domain.

4.4.1 GUI description

Table 1 shows examples of the type of information about the GUI that the tool should obtain.

In a similar way to the way HTML describes the layout of

web pages, XML or other scripting languages can be used to fully describe an HDA's interface. The tool should include a parser to interpret such scripts and extract relevant information for analysis.

Control / component	Properties
Button	Width, height, text font type and size, position, enabled/disabled, visible/not visible
Table	Width, height, position, number of columns, width of each column, data type for each column, test font type and size, enabled/disabled, visible/not visible
Menu	Number of items, greatest depth (number of levels)
Dropdown list	Width, height, position, number of items

Table 1: GUI information examples

Such a script may be written manually (albeit tediously), or alternatively the tool should be able to generate it automatically using reflection on compiled code, in which case the necessary information can be extracted without the need for an intermediary script or a parser. However, a human-readable representation of the GUI, such as in the form of an XML document, could well be useful for quick manual review or knowledge transfer. Additionally, a script description allows for evaluation of GUI designs that are yet to be implemented in code.

4.4.2 Recording user interaction sequences

The conventional method of capturing how a user interacts with an HDA is to observe and video/audio-record them using the HDA or an equivalent working prototype. The time and resources required for recording and subsequent analysis of the collected data makes this one of the most costly processes in a usability evaluation. Also, most tape mediums are adequate for displaying images of people, but lack resolution capability to do so for images of computer displays. Furthermore, test users are also often distracted or intimidated by the presence of observers, cameras and other recording equipment, and do not perform as they normally would. An alternative recording procedure is clearly necessary.

Lafleur (2001) describes an effective recording alternative in the form of Direct-Digital Recording (DDR) software, which operates within the computer running the AUT. An equivalent should be practicable for use on a handheld device, though additional care should be taken to ensure that the DDR software does not affect the performance of the AUT, particularly given the smaller processing power of handhelds'. Because such DDR software is likely to interact directly with the AUT at code level, it should give developers greater flexibility in customizing the recording process. For example, as with conventional, serial tape systems the footage has to be divided manually into clips corresponding to separate tasks, a DDR system should allow developers to programmatically specify which actions (or sequence of actions) identify the beginning or end of a task, or to ignore or highlight particular actions that the user performed (such as ignore scrolling, and highlight when the Help button is clicked).

Similarly to using scripts to describe a GUI, documenting action sequences using a scripting language such as XML would provide a readable and editable representation of the data, allowing developers again to design action sequences for evaluation without the need for working prototypes. The schema should be comprehensive to allow most if not all action types to be described, as well as their properties and the details of how the actions relate to each other, such as:

- Ordered and unordered collection of actions
- Any actions to be ignored
- Multiplicity / frequency of actions

A well structured schema would allow portability so that the data can be used with different applications on different platforms.

4.4.3 User behaviour and comments

The conventional recording practices by video provide useful footage of user behaviour. Body language gives a good indication of the user's thoughts and feelings as they attempt the tasks; frowning, pauses, uncertain actions and other expressions of confusion or frustration strongly suggest poor usability. Comments and thoughts of the users themselves are also of course a good insight into how usable the users found the AUT. Although these tasks would undoubtedly be a huge challenge to fully automate, they should still be included in the usability evaluation of HDAs according to Lee and Grice (2004) and thus the tool should aim to better support them.

4.4.4 Task definitions

Providing some way for developers to input task definitions would be a big challenge, since these range broadly in scope and can be generic or very specific to a particular HDA. For the purposes of templates or style guides however, it should be sufficient to classify them in a more generic sense. For example, the case study in Section 2.1.1 may be classified a 'data input form', the case study in Section 2.1.2 a 'data view and selection form', and the case study in Section 2.1.3 a 'text and image display' form.

4.5 Results presentation

To encourage the awareness of usability throughout the development team, beyond just usability testing specialists, the output of the tool should be meaningful and useful to people in different roles and with varying technical knowledge. For example, developers and testing specialists would want to view both aggregate and detailed statistics for analysis results, though code-level references would be more relevant for the former than the

latter. Managers and sales personnel would find high level aggregation statistics rather than detailed figures more useful, and would likely be more interested in qualitative aspects such as users' feedback on aesthetics and overall satisfaction. The tool should thus present results from multiple perspectives and in ranging levels of detail.

Most importantly though, is that the output is quick and easy to interpret. This is essential to speed up the evaluation process and therefore make it practical to perform on an iterative basis. Graphical visualizations such as graphs and other diagrams should be used where possible, and critical issues emphasized to draw attention. There should also be a clear indication of the next step forward, in the way of modification suggestions to reduce the number of warnings or test failures.

Finally, analysis output should be available in a range of formats, so that it can be processed by other applications such as for printing, archiving or further analysis.

5 HUIA testing framework

This section describes the Handheld device User Interface Analysis (HUIA) testing framework, an automated usability testing tool prototype that we have developed. HUIA is described in more detail in Baker et al. (2006). The tool was designed to meet some of the functional requirements discussed above.

The following sections describe the main functionalities of the HUIA testing framework.

5.1 Actions recorder and editor

The Recorder is deployed on the handheld device running the AUT. Its implementation is centered on its registration as a listener on the components of the form, allowing it to keep track of events and the associated components as they happen (such as clicks, scrolls and drags). This information is recorded in an XML document, an Actual Action Script (AAS), as shown in Figure 4.

```
<input control="textBox_lastName" value=
"das" time="24" type= "TextBox"/>

<click control="radioButton_male" value=
"True" time="25" type= "RadioButton"/>

<click control="button_update" time="30"
type= "Button"/>
```

Figure 4: AAS example

The recorder concept is similar to that in the EDEM system, which collects usage data to aid the development process as described in Hilbert et al. (1998). The Recorder is lightweight and operates behind the scene, so that the performance of the AUT is unaffected, and the test user is likely to interact with the AUT as they would normally. Currently, the Recorder captures only a subset of user interactions; in particular, scrolling is not able to be recorded because it is not currently supported by .NET Compact Framework 2.0.

An Expected Action Script (EAS) is an extension to the AAS, documenting the interactions with a given form as performed by the developer, and thus describes the intended usage of the interface.

Figure 5 shows an example portion of an EAS:

```
<input control="textBox_firstName"
value= "John" type= "TextBox"
mult="1"/>

<click control="radioButton_male"
value= "True" type= "RadioButton"
```

Figure 5: EAS example

The EAS also documents other information for use in a comparison analysis (see *Section 5.2 Comparison analysis*), such as actions to ignore, optional actions and ordering of actions, since in reality there may be different ways of achieving the same task.

There are currently two ways of creating and editing an EAS. Firstly, developers can use the tool's built-in EAS editor, where the form is loaded and displayed, and the EAS created as the developer interacts with the form. The GUI of the editor also allows developers to specify the various EAS properties. Alternatively, developers can create and edit EASs as XML files directly, using any text editor.

5.2 Comparison analysis

This analysis aims to evaluate some aspects of navigational burden by comparing, for a given task, how the developer expects a form to be used with how a test user actually uses it, and highlights the differences between the two. Specifically, an algorithm attempts to match an AAS to a specific EAS. More than one AAS can be compared with a single EAS, such as in the case of having multiple test users attempting the same task.

Figure 6 presents a screenshot of a set of comparison results.

5.3 Assertions analysis

Assertions are essentially checks that evaluate to either true or false, and the tool's assertion analysis involves value assertions on a collection of usability metrics, based on upper and lower threshold values. Developers may use the tool's set of default threshold values, or specify their own to better suit the particular application. Each type of assertion can also be disabled independently of the others, and assertion settings are preserved across tests or iterations. Assertion results are presented in tables with rows color coded depending on the result, in a similar fashion to those of existing unit testing frameworks, such as JUnit. Three types of assertions exist, as described in the following sections.

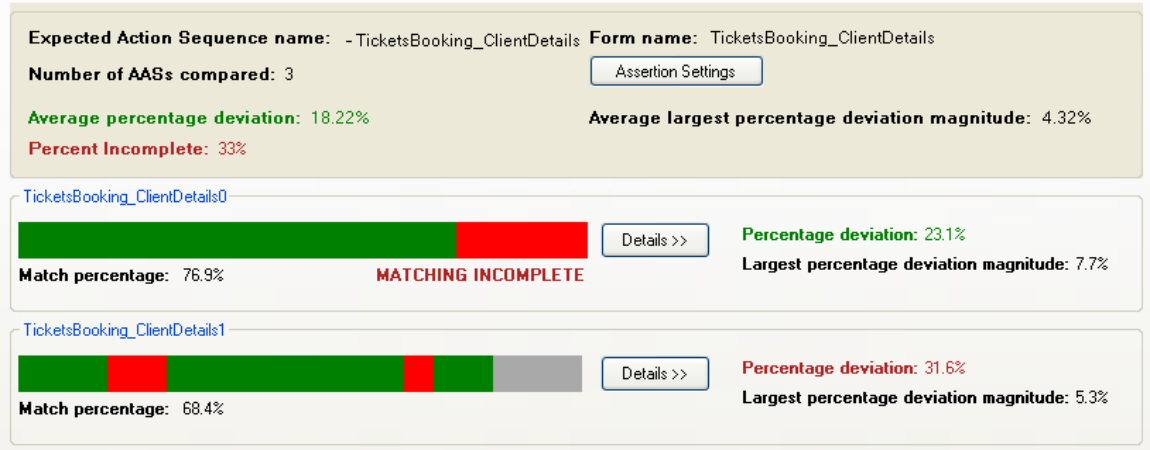

Figure 6: Comparison analysis results

5.3.1 Form assertions

These assertions are performed on the compiled .NET forms of the loaded assembly, and are concerned with the static properties of a form, discussed in Section 4.1 *GUI Evaluation*. For example, an HDA can be asserted to ensure that the maximum number of font types is not exceeded, or that a form has the minimum percentage of white space.

5.3.2 Action script assertions

Action script assertions are performed on EAS and AAS descriptions, and assess the usage of a form by asserting that the amount of each type of user interaction falls within an acceptable range. Examples of threshold values that an action script can be asserted against include:

- Maximum number of invocations per component (for example, clicks, text inputs).
- Maximum time taken/passed.

5.3.3 Comparison result assertions

Comparison result assertions are performed on the summary statistics for comparisons, which include:

- **Percentage deviation:** The percentage of actions that were deviations, or mistakes.
- **Largest percentage deviation magnitude:** The size (as percentage of total number of actions) of the largest deviation.
- **Percentage of incomplete comparisons:** The percentage of users that failed to complete the task.

Both the action script and comparison results assertions evaluate the HDA's navigational burden.

5.4 Hotspots analysis

This analysis shows the frequency of use for each component on a form for a given task; the form is displayed with the components color coded according to the amount of activity they received for a particular interaction type, for example, the number of clicks a component received.

Figure 7 shows part of a screenshot for the Hotspot analysis. The more intense the red, the greater the amount of activity a component received for the given interaction type. Uncolored or white components are those that received no activity. This analysis also examines the HDA's navigational burden.

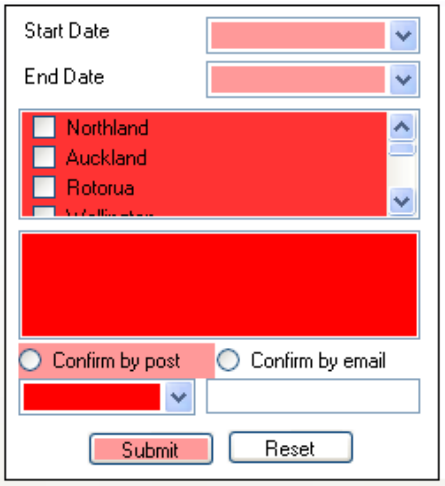

Figure 7: Hotspots analysis results

6 Related work

In this section, the effectiveness of some existing software designed to automate or improve the efficiency of usability testing is examined. The various tools discussed are typical of the types of tools currently available commercially.

6.1 Tool descriptions

6.1.1 Morae

Morae is a commercial usability testing software package produced by TechSmith, and consists of four modules. The *Recorder* is installed on the test user's machine and records screen and system activity; the *Remote Viewer* allows testers to remotely control the *Recorder* and to view, hear and annotate the recordings in real time; the *Manager Analysis* module allows testers to isolate segments of the recording, and

provides a search editor to quickly locate user actions. Metrics such as time on task, number of clicks or pages viewed and delay times are also automatically calculated, based on time-stamped and indexed events on the video; the *Manager Presentation* module is used to edit, annotate (textual or audio) and title video clips, for use in Morae or in other programs (such as Microsoft PowerPoint), and optionally with videos not recorded with Morae.

In many respects, Morae successfully automates and significantly reduces the cost of conducting and analyzing a user test by facilitating more efficient collection and usage of recorded data. Although some calculation of usability metrics is provided, the core of the package is still very much concerned with the making and manipulation of recordings, with the tester left responsible for the bulk of the analysis and interpretation of data. For example, the tester must still sift through the recordings and decide which parts to highlight and further analyze, or must determine the significance of twenty mouse clicks compared to fifteen.

Furthermore, Morae is generally intended for use towards the end of the development process with a working prototype or even the completed product; making appropriate modifications following the test at this stage is often more time-consuming and labour intensive than if they were performed during development.

6.1.2 Web based user testing tool

Bailey and Bailey (2003) describe a usability testing tool that facilitates user testing of websites over the internet. Special software is installed on the test user's computer, to facilitate the setup of tests, post-test software removal, data collection (such as questionnaire answers, typed input, links navigated, user comments, and times for clicking, thinking, page loading) and transmission of results to a central server for analysis. The tool also generates reports that comply with the *Common Industry Format for Usability Test Report* (v2.0, May 2001).

This tool is also successful in automating some processes in user testing, and has the advantage of being online, allowing test users to download the required software. However, it does not capture user behaviour, which is an important output of user testing, and like Morae still requires the tester to perform the bulk of the evaluation.

6.1.3 Watchfire Bobby (WebXACT)

Watchfire Bobby is a very commonly used web accessibility testing tool, of which a restricted version is available for free use online as WebXACT. Bobby traverses a website (both local pages as well as web pages behind a firewall) and checks if each page meets various accessibility requirements, such as '*readability by screen readers, the provision of text equivalents for all images, animated elements, audio and video displays'*. The checks are based on an assessment of the website's HTML against a set of accessibility guidelines, including Section 508 of the US Rehabilitation Act and the W3C's Web Content Accessibility Guidelines (WCAG).

Following a scan, results are presented in tabular form, indicating passes, warnings, failures, and errors for the different requirements. General properties and a metadata summary for the pages are also presented.

Bobby is much more lightweight and quick to use than tools such as Morae, and could easily be incorporated into earlier stages of the development process. However, it focuses on quantitative and other static data, with virtually no attention paid to how the website's interface is actually used.

6.2 Tool effectiveness

Table 2 describes how well each tool meets the functional requirements discussed in Section 4 *Functional requirements*. The legend used for the table follows:

M = Morae ✓ = Achieved

WB = Web based tool o = Achieved to some extent

WX = WebXACT × = Not achieved

H = HUIA framework

Criteria	M	WB	WX	H
GUI description input	×	×	o	✓
User actions input (recorder)	✓	✓	×	✓
User behaviour and comments input	✓	×	×	×
Task definitions input	×	×	×	×
GUI evaluation	×	×	✓	✓
Navigational burden analysis: form layout	×	×	×	O
Navigational burden analysis: user actions	×	×		O
User behaviour and comments analysis	O	O	×	×
Patterns and templates	×	×	×	×
Results presentation	✓	✓	o	✓

Table 2: Effectiveness of existing tools

It can be seen that both Morae and the web based testing tool focus primarily on automating some processes of an actual user test, but neither attempt to perform any evaluations. This means that testers must still perform this work themselves. WebXACT assesses the GUI, but does not take into account other

aspects. The HUIA testing framework on the other hand attempts to evaluate most aspects of usability.

It should be noted however that none of these tools capture the description of tasks that the HDA was intended to support, nor provide patterns of proven paradigms that promote good usability.

7 Conclusions

Usability is very important for handheld device applications, because those with poor usability can lower the productivity of its users and incur costs for businesses, thus undermining the value of a mobile business solution. Automating aspects of usability testing can improve testing efficiency and better facilitate its integration with the development process. Ideally, an automated usability testing tool should capture a range of inputs, perform analyses on different aspects of usability, present results clearly, be simple and flexible to use, and able to be used throughout development.

None of the existing tools discussed meet all the requirements described. Notably however, none of the tools are able to suggest good usability solutions, they can only perform evaluations. Of the tools discussed, the HUIA testing framework addresses most requirements, though still requires future development, which may include GUI consistency checking such as that outlined in Mahajan et al. (1997), and the inclusion of standard interface description languages such as those described in Souchon et al. (2003).

8 References

Ahmad, R.; Zhang Li; Azam, F. (2006): *Measuring Navigational Burden*. Fourth International Conference on Software Engineering Research, Management and Applications, pp.307 – 314.

Bailey, R; Bailey, K (2003): *Expediting the Usability Testing Process*. 13th Annual Conference of the Usability Professionals' Association.

Baker, S; Au, F; Warren, I; Dobbie, G. (2007): *HUIA: A Tool for Automated Usability Testing*, UoA-SE-2007-1, University of Auckland.

Blewitt, A.; Bundy, A.; Stark, I. (2005): *Automatic Verification of Design Patterns in Java*. 20th IEEE/ACM International Conference on Automated Software Engineering, pp. 224 – 231.

Hilbert, D., Robbins, J., Redmiles, D. (1998): *EDEM: Intelligent Agents for Collecting Usage Data and Increasing User Involvement in Development*, International Conference on Intelligent User Interfaces, ACM Press, pp. 73-76.

Institute of Electrical and Electronics Engineers (1990). IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY.

Ivory, M. and Hearst, M. (2001): *The State of the Art in Automated Usability Evaluation of User Interfaces*, ACM Computing Surveys, 33 (4), pp. 173-197.

Kane, D. (2003): *Finding a place for discount usability engineering in agile development: throwing down the gauntlet*. IEEE Proceedings of the Agile Development Conference, pp. 40 – 46.

Kurosu, M.; Kashimura, K. (1995): *Determinants of the apparent usability*. IEEE International Conference on Systems, Man and Cybernetics, Vol. 2, pp. 1509 – 1514.

Lafleur, A. (2001): *Camera-less Video: Usability Test Recording and Presentation in Direct Digital Form*. Tenth Annual Conference of Usability Professionals' Association.

Le Peuple, J, Scane, R. (2003): *User Interface Design*. Crucial, a division of Learning Matters Ltd.

Lee, K; Grice, R. (2004): *Developing a New Usability Testing Method for Mobile Devices*. Professional Communication Conference, pp. 115 – 127.

Mahajan, R. and Shneiderman, B. (1997): *Visual and Textual Consistency Checking Tools for Graphical User Interfaces*. IEEE Trans. Softw. Eng. 23(11), pp. 722-735.

Moe, K; Dwolatzky, B; Olst, R. (2004): *Designing a Usable Mobile Application for Field Data Collection*. IEEE AFRICON. Pp. 1187 – 1192.

Morae: Usability Testing for Software and Websites. http://www.techsmith.com/morae.asp, visited 8th March, 2006.

Nielsen, J. (1993): *Usability Engineering*. Academic Press, Inc.

Patton, R. (2006): *Software Testing*. 2nd edition. Sams Publishing.

Scholtz, J. (2006); *Usability Evaluation*. http://www.itl.nist.gov/iad/IApapers/2004/Usability%20Evaluation_rev1.pdf, visited 9th March.

Nathalie Souchon, Jean Vanderdonckt (2003): *A Review of XML-compliant User Interface Description Languages*. 10th International Workshop on Interactive Systems. Design, Specification, and Verification, pages 377-391.

Spolsky, J. (2001): *User Interface Design for Programmers*. Apress. New York, NY.

Watchfire: Accessibility Testing. http://www.watchfire.com/products/webxm/bobby.aspx, visited 8th March, 2006.

Weiss, S. (2005): *Handheld Usability: Design, Prototyping, & Usability Testing for Mobile Phones*. Proceedings of the 7th Conference on Human-Computer Interaction with Mobile Devices and Services.