

OTDK-DOLGOZAT

Szóke Gábor, Muhi Kornél
MSc

2013

A Semi-Automatic Usability Evaluation Framework

Fél-automatikus használhatóság kiértékelő keretrendszer

Gábor Szőke, Kornél Muhi

Kézirat lezárva: 2012. december. 30.

Abstract

Most of the software maintenance costs come from usability bugs reported after the release and deployment. A usability bug is really subjective, hence there is a large communication overhead between the end user and the developer. Moreover, the reputation of the software development company could be decreased as well. Therefore, proactively testing and maintaining software systems from a usability point of view is unambiguously beneficial.

In this paper we propose a research prototype, the Usability Evaluation Framework. The framework is based on a well-defined usability model, it calculates usability metrics, it integrates questionnaires and it also ensures several meaningful reports. We have successfully applied the framework to evaluate and to improve the usability of two industrial software systems. These software systems were studied in working production environments.

Contents

Abstract	1
1 Introduction (Muhi)	2
2 Background (Muhi, Szőke)	4
2.1 Project background	4
2.2 Requirements	5
3 Usability model (Szőke)	7
4 Framework (Szőke)	10
4.1 Architecture	10
4.2 Input format of the framework	12
4.3 Report	13
4.4 User interface	14
4.4.1 Log uploading	14
4.4.2 Metrics calculation	14
4.4.3 Survey handling	14
4.4.4 Report generator	15
4.5 General application guidelines	15
5 Case study (Szőke, Muhi)	20
5.1 Integration and operation	20
5.2 Results	20
5.3 Improvement of the systems	24
6 Evaluation (Muhi)	26
7 Related work (Muhi)	27
8 Conclusions (Szőke)	29
Acknowledgement	30

Chapter 1

Introduction

A popular belief about software maintenance is that it primarily involves development of new features, testing, and fixing of programming bugs (e.g. a null pointer exception). Although, some researchers show that a significant part of software bugs are related to some kind of usability problems of the investigated applications. For example, Grubb and Takang point out [1] that usability is a major indispensable premise for software maintenance:

"If a system for any reason is not used, it may as well not exist. Maintenance is only an issue for a system that is used and that evolves with use."

Furthermore, Landauer [2] reports an interesting distribution of software bugs and draw attention to the importance of usability bugs:

"About 80 percent of software life cycle costs occur not during development but during the maintenance period. In turn, 80 percent of these maintenance costs are a result of problems users have with what the system does, not programming bugs."

There are some prevalent methods to evaluate and test usability. Usability testing in labs is really useful because it can recover serious usability bugs. However, it is also expensive and time consuming because real users have to leave their current tasks and they have to go the usability lab. It also requires experienced moderators and human factor experts, otherwise they probably face communication problems with the users. Furthermore, a usability lab is just a simulation of a real production environment, therefore some usability problems could be missed. However, remote usability testing is a really promising research direction, most of the problems still remain unsolved in this case as well.

Several usability testing tools are available, but these work only on one field, typically on web pages. To the best of our knowledge, there is no general solution available that supports the usability evaluation of desktop, mobile and web applications as well.

In this paper, we present a research prototype, the Usability Evaluation Framework (UEF) that supports usability testing and usability maintenance in real production environments and can be applied on several application fields (e.g. web, desktop and other) as well. We have two goals with the framework. Principally, we want to test and improve the usability of two industrial software systems. Subsidiary, we want to develop a completely general framework that is widely applicable.

The paper is organized as follows. In the next section, we describe the background of the project. In Section 3 we introduce the key component of the framework, a general usability model. Next, we describe the details of the framework in Section 4. Section 5 shows our results about applying the framework in case of two industrial software systems. Next, we evaluate the Framework based on the requirements. Finally, the last sections present related works and conclusions.

Chapter 2

Background

2.1 Project background

Integrated collection management systems (ICMS) manage the business workflow of libraries and cultural institutes. These systems tend to be divided into multiple modules based on the different workflows they provide. Libraries can choose which module to include in their deployment of the system. These modules can be cataloging, circulation, stock management, financial transactions and public access modules. For example, library catalog modules usually handle the metadata associated with the works contained in the institution, containing several layers of aggregated metadata in a standard-conformant manner. The complexity of collection workflows and the functions of the software supporting them require a complex approach to these problems. Since these software products are essential to most of the work done in these institutions, the ease of use and ergonomics of these systems are crucial.

As a management organization, DEAK Plc. helps our institute and the Biological Research Center of the Hungarian Academy of Sciences of our hometown in the industrial and economic utilization of their existing research capacities and research results. Monguz Ltd. is one of the leading software development companies in Hungary in the field of ICMS. Monguz Ltd. assigns us to test, to evaluate and to improve the usability of two ICMS systems. The usability evaluations were performed in the daily operative work of two selected Hungarian libraries with the help of the Usability Evaluation Framework (UEF).

2.2 Requirements

To fulfill the claims of Monguz Ltd. and to ensure that UEF can be applied widely in the future we set up the following requirements.

Evaluation validity

The usability evaluation is required to be valid as possible. The rationale behind this requirement is to ensure that the results of usability evaluation can be trusted. We have defined four subrequirements:

- *Production environments.* Usability testing and evaluation are typically executed in a testing or staging environment, e.g. in a usability lab, where a moderator controls and observes the representative users delegated by the customer. Nevertheless, several usability bugs can be detected with this technique, and the laboratory circumstances also determine missed usability bugs. For example, customer delegates only one or two users, the moderator can affect the behavior of the user, and so on. Therefore, the evaluation should be performed in real-life working *production environments*.
- *Detect red routes.* Red routes are the most important roads that can be a potential bottleneck for a traffic jam. The flow of traffic has to be ensured as much as possible in the case of such red routes. Red routes can be adapted to the UI of a software system as typical sequences of user actions that are crucial considering certain indicators [3].
- *Subjective.* Usability is a really subjective quality. Therefore, the solution has to support subjective evaluation as well.
- *Concrete bugs.* Of course, information about concrete usability bugs is substantial. For example, it can be a frequent and complex user interaction sequence (i.e. a red route) that could be re-engineered as a much usable and straightforward wizard; it can be a complex form that is really difficult to be filled out, therefore it is time-consuming for the users to work with, and so on.

Transparency

It is required that the evaluation should not affect the daily operational work of the users. The rationale behind this requirement is to protect the customers of Monguz Ltd. If users suffer from any kind of interruption or disturbing factor (e.g. video analysis) then

unacceptable extra costs are generated from a business point of view. In this case we have defined only one subrequirement, *silent data logging*, i.e. the framework cannot disturb the users in their daily operational work.

Automatic

UEF has to be automatized as much as possible. The rationale behind this requirement is to reduce extra costs generated by unnecessary manual work.

Domain-independent

UEF should be easily applicable in later projects outside the domain of ICMS. The rationale behind this requirement is to support the potential application of UEF in other projects as well. It can be ensured by a domain independent and well-defined *usability model*. *Usability model* is responsible (1) to define a schema, and, (2) to store usability related data as general as possible.

Application-independent

UEF should be applicable in case of desktop, web or mobile applications as well. The rationale behind this requirement is to support the potential application of UEF in other projects as well. It can be ensured by an application independent and well-defined *general data interface*. Naturally, application specific data logging cannot be generalized, it has to be specific to the investigated application.

Platform-independent

UEF should support Windows, Linux and other operating systems as much as possible. The rationale behind this requirement is to support the potential application of UEF in other projects as well.

Chapter 3

Usability model

A well defined data model (shown on Figure 3.1) is the key to our approach, and the basis of the other components of the UEF framework. In this section we introduce our model.

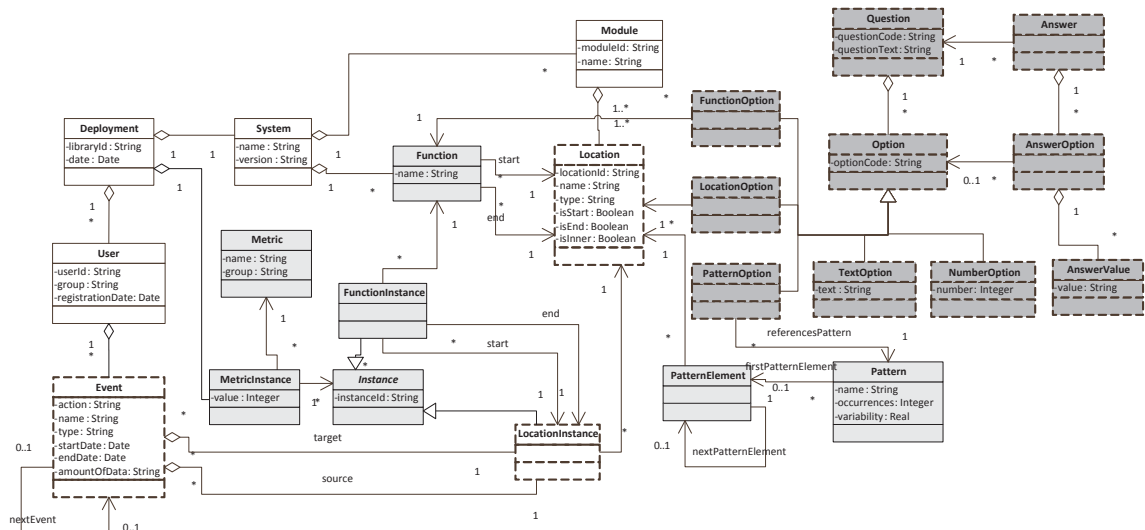


Figure 3.1: Usability model

System nodes

are denoted by non filled nodes with continuous borders in Figure 3.1. The software vendors usually deploy a customized build of the application to the customers to match their special needs. Based on this observation, the *Deployment* entity has a central role in the model, and it contains the *System* entity. Furthermore, it contains only those *Module* entities that the users have actually in the current deployment. Deployment also contains the registered *Users*.

Low level nodes

are denoted by non filled nodes with dashed borders in Figure 3.1. Whenever a user clicks on a button in the application or presses a key on the keyboard, an event will be generated. Such activities are represented by the *Event* entity. It stores the event's creation and completion date, the name, the type, the executed action, and (optionally) the size of the processed data of the executed action. Besides storing such basic event data it is also required to store the source and target locations too. The source indicates where the event has been triggered (usually a menu item or window). The target represents the result of the event. Source and target information are represented through *Location* and *LocationInstance*. The *Location* entity represents the abstraction of a certain location with its name, type and other information. *LocationInstance* represents a concrete instance of a *Location*. For example, in the studied ICMS applications it is possible to open multiple user profile windows. Such windows have to be handled together in some point of view, and that is the reason behind *Location*. While in other aspects, we have to be able to distinguish them, and that is the reason behind *LocationInstance*. An event is generated by a user, hence *User* contains the corresponding *Events* in the model. The events are stored chronologically, which is represented by the *nextEvent* relation.

Derived nodes

are typically based on low level nodes, and denoted by filled nodes with continuous borders in Figure 3.1. A functionality of the system can be described by a sequence of events between two locations. The model represents such functionalities with the *Function* entity. The *FunctionInstance* entity is a concrete instance of the referenced *Function*. It points to two *LocationInstances* according to the *Location* types defined by the *Function*. *Metric* represents the calculated properties of functions and locations. The *MetricInstance* entities assign concrete values to an instance (*FunctionInstance* or *LocationInstance*) based on the referenced metric. Because the metric entities are the same in case of every deployment, it is enough to store them just once.

Frequent user interaction sequences (i.e. patterns of user actions) are also represented in the model. A pattern consists of two parts: a main entity (*Pattern*), which represents the sequence's generated name, occurrence and variability; and *PatternElements* which links together the sequence's pieces. A pattern entity points to its first element, and then a *PatternElement* refers the next element recursively.

Survey nodes

are denoted by filled nodes with dashed borders in Figure 3.1. The model is also capable of storing data for generating online questionnaire, and for representing the results of the questionnaire. A survey usually contains a few *Questions* which have more *Options*. The user has to pick one or more of these options. An *Option* can be any kind of a field: a simple text (*TextOption*), a number (*NumberOption*) or a reference field to a function (*FunctionOption*) or location (*LocationOption*). The last two come in handy when we want to ask the users about the investigated system and about the automatically detected information (i.e. automatically calculated information can be related with subjective user opinions). Answer stores the options picked by the user (*AnswerOption*). The *Answer-Value* is introduced to store comments and justifications about the answers. Patterns can also be referenced in the survey (*PatternOption*).

Framework

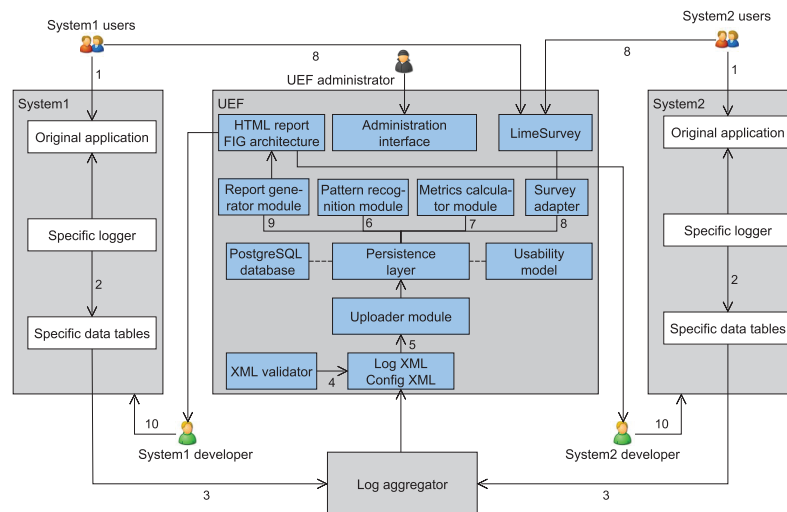


Figure 4.1: Architecture of the UEF framework

In this section we introduce the Usability Evaluation Framework (UEF). First, we briefly introduce the concrete steps of the framework. Afterwards, we show the input format and the generated report. Next, we describe general application guidelines of the framework.

4.1 Architecture

In the followings, we shortly introduce the architecture of the framework (see Figure 4.1). The arrows and numbers represent the steps in chronological order.

The first three steps are specific to the current project and to the ICMS domain. We also show these steps to give a wider picture about the framework and its surroundings,

and to ease the understanding of the case study (Section 5). Each step has been designed and developed to ensure the previously defined requirements.

We fulfill the *Transparency / Silent (specific) data logging* requirement by executing the following steps:

1. At the first step the users of the library's software system - the readers and librarians - use the *original application* and perform some events and actions.
2. We have developed *specific logger* components to collect usage information about two different library collection management systems. The logging mechanism is tuned to catch events significant to the user workflow, such as moving between windows or client-server messages. The logger components collect usage information in *specific data tables*.
3. The specific loggers collect information in different physical locations. Therefore, we developed the *log aggregator* component to collect the different logs from different locations uniformly (step three on Figure 4.1). At the end, this component exports the collected information into the input format of the UEF.

The input of the UEF is a well defined XML format (see Section 4.2 for details). It fulfills the *D/A independent / general data interface* requirement (Log XML and Config XML on Figure 4.1).

4. The XML validator component ensures the correctness of the data, so further analysis steps can trust the XML inputs. In the current project it validates files produced by the log aggregator. There are two kinds of log-files. *Config XML* describes meta information about the system, while *Log XML* contains the concrete usage data. First, the validator checks the syntax of the logfiles according to the predefined XSD files. Second, the validator analyses the files semantically, i.e. it checks complex rules that cannot be described in XSD.
5. The *uploader module* uploads the logged data into a persistence framework for further processing. The persistence layer ensures data abstraction above a PostgreSQL database, and the developers have to give only an UML model (as its schema). The schema of the persistence layer is configured with the *usability data model* (see Section 3).

The usability model fulfills the *D/A-independent / general data model* requirement. This model is the base of the other analyser components that actually fulfills the *Evaluation validity* requirements:

6. To process the user events stored in the persistence framework, a sequence recognition module has been developed based on suffix-array representation. This module recognizes the frequent sequences of user interactions (patterns), i.e. it searches for hidden functionality and *red routes* in the applications. It has two parameters which affects the results, the minimal length and the minimal occurrences of a sequence. The result can be useful as a starting point to develop new wizards that can improve the usability of the system.
7. The metrics calculator module can calculate several numeric or textual properties for further analysis. This module can be extended, calculation of new metrics requires only a new metric calculator plugin to be developed.
8. Surveys collect subjective opinions from the users. These surveys can contain questions regarding the results of the current data analysis. This module is capable of generating configuration files which can be imported into the LimeSurvey open source survey engine. These surveys were filled out by the users of the subject systems, and the results have been uploaded back into the persistence system for further processing. Questions can be easily added, modified and removed in the framework.
9. Report generation: Based on the collected information (metrics, patterns and survey results) the UEF framework generates a HTML report. Besides the report, user interactions can be visualized based on the frequency of the executed actions.

At the final 10th step the subject systems were optimized in usability aspects based on the generated report.

4.2 Input format of the framework

The general, domain- and application-independent data interface of the UEF is represented by well-specified XML formats. The development of these XML formats is heavily influenced by the usability model.

The interface consists of two XML documents: the *configuration descriptor* document (Config XML in Figure 4.1), and the *events log descriptor* document (Log XML in Figure 4.1).

The configuration descriptor contains general meta-information about the investigated software system. An example configuration descriptor XML file is shown in Figure 4.2.

The events log descriptor document stores information about the users' activities. The format of the events descriptor log files is very simple: the only meta-information it re-

quires is the deployment identifier while every other information concern the user events. A user event consists of a name (e.g. "search for readers"), an action (e.g.: "open window") and the source type (e.g.: "menu"). The user event also contains the user who performed the event, and a source (e.g. "File menu") and a target location ("Save dialog") where the event begins and ends. The source and target locations are given by their ids which refer to the configuration descriptor file. An example event log descriptor XML file is shown in Figure 4.3.

The reason of the separate system descriptor is to make the events descriptor less redundant. Altogether, these XML files contain the data that is needed by our model.

4.3 Report

A report generator module was also developed in the UEF. It requires the execution of the analyzer components, i.e. metrics and patterns have to be calculated and stored in the persistence layer, and the questionnaire results have to be also stored there.

System level calculations are presented in the first section of the report:

- The report starts with the list of the unused windows. These windows are available in the current deployment, but users have not opened them during the examination period. These can help to detect those parts of the application that have not been used by the users at all.
- The second table consists of the metrics calculated for different location types. These are the aggregated average/minimum/maximum values of the corresponding instance metrics.
- The third table collects metrics about the functionalities (i.e. predefined Location sequences) of the system. They tell us the starting point of a functionality with the corresponding metrics.
- The last table of the section presents the patterns recognized from the user actions. The patterns are ordered by their importance. Importance is determined by the linear combination of frequency and length of the patterns. An additional page has been created for every pattern, which visualizes the pattern in the form of a map of locations. An additional column shows that how many percent of the users think that a wizard should be created for this particular pattern.

The second section of the report contains results about the questionnaire. The framework ensures to setup arbitrary questionnaires. As the last mark of the survey section, there is a link to a page that summarizes the opinions and personal experiences of the

users. The report generation is based on templates (see [4]). An example report in Hungarian language is shown on Figure 4.4.

4.4 User interface

In order to gather all the features easily accessible by the administrator of the AUEF, a graphical user interface has been also developed. An example screenshot of this application is shown on the Figure 4.5. It can be divided into four tabs: Log uploading, Metrics calculation, Survey handling and Report generator. Each of these tabs provides possibility of setting the parameters of the selected function.

4.4.1 Log uploading

Here, log files can be uploaded. Before uploading, it is required to perform the validation step. After the log files have been chosen, the validation process can be initiated by clicking on the corresponding button. If either of the files are not valid, fouls are shown in an additional window. After uploading the log files, a notification message will appear regarding the success of the action. By clicking on the OK button, the user will be automatically directed to the second tab.

4.4.2 Metrics calculation

This tab is responsible for metrics calculation and pattern detection. The settings consist of giving the minimum length of a potential sequence, and the minimum number of occurrences of the same sequence. We have defined these values as default 3 and 3, respectively. After successful metric calculation and pattern detection, the third tab awaits the user.

4.4.3 Survey handling

Here, survey question generation and the answers given by the participants can be managed. This tab requires four parameters: the *survey file*, the *survey template file*, the *generated survey path*, and the *answer file path*. The survey file is an XML file taken from the LimeSurvey engine containing the general questions. The survey template file is a special file containing special markers for the identified patterns. This template file is filled with the actual pattern data, and afterwards it can be uploaded back into the LimeSurvey engine. The generated survey path states where the generated survey file should be put after processing. Survey results can be obtained from LimeSurvey in a CSV file format.

The answers file path parameter should point to this file. After this, the answers can be uploaded into the persistence layer.

4.4.4 Report generator

The last tab consists of the report generator settings. An output folder has to be given. The report contents can be specified: the user can select whether the report include patterns or survey results besides the calculated metric values. There is an other option to generate visualization graphs from the user interactions regarding the window instances or the aggregated ones.

4.5 General application guidelines

We have also developed a template that ensures the application of the UEF prototype in arbitrary domains as well.

1. **Generating UEF's XML inputs.** Either the XML files are directly created during the logging or if the investigated system logs elsewhere (e.g. to a database table) then the information have to be converted into UEF's XML files.
2. **Data collection (in production environment).** This step consists of the actual use of the observed systems, during which the logger components collect the user interaction data into XML files. Depending on the amount of data you want to collect, this could take weeks or months.
3. **Surveying questionnaires.** A specified questionnaire will be generated as the metrics and the sequences have been calculated. This survey will be presented to the users of the observed systems. Depending on the number of the users, filling out the survey takes 1 or 2 days.
4. **Automatically generating reports.** The framework generates an HTML report based on the calculated information and the opinions given by the users. This step takes maximum 1 or 2 hours depending on the amount of data.

Apart from the fact that the survey has to be filled by the users, the UEF is automatic.

```

1 <deployment libraryId="library1" date="2011-11-30"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:noNamespaceSchemaLocation="extformat.xsd">
4   <system name="Big Library" version="1.0.0"
5     deploymentDate="2005-01-01">
6     <module id="module01" name="reader"/>
7     <module id="module02" name="operator"/>
8   </system>
9   <locations>
10    <location locationId="renting.books" name="Select books to
11      rent" type="window" module="operator"/>
12    <location locationId="renting.reader" name="Data of the
13      reader" type="window" module="operator"/>
14    <location locationId="renting.summary" name="Summary about
15      the rent" type="window" module="operator"/>
16    <location locationId="renting.help" name="Renting books help"
17      type="help" module="operator"/>
18    <location locationId="renting.fault" name="Renting books
19      error" type="error" module="operator"/>
20  </locations>
21  <functions>
22    <function name="Rent a book" startLocationId="renting.books"
23      endLocationId="renting.summary"/>
24  </functions>
25  <users>
26    <user id="1" group="administrator"
27      registrationDate="2011-11-30"/>
28    <user id="2" group="operator" registrationDate="2011-11-30"/>
29  </users>
30</deployment>

```

Figure 4.2: Example for the configuration descriptor

```

1 <log xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:noNamespaceSchemaLocation="logformat.xsd"
3   libraryId="library1" date="2011-11-30">
4   <events>
5     <event action="rent" name="" type="button"
6       startDate="2011-11-27T08:03:47.103"
7       endDate="2011-11-27T08:03:47.103">
8       <source locationId="main" instanceId="1"/>
9       <target locationId="renting.books" instanceId="2"/>
10      <user id="1"/>
11    </event>
12    <event action="rent.reader" name="" type="button"
13      startDate="2011-11-27T08:03:52.163"
14      endDate="2011-11-27T08:03:52.163">
15      <source locationId="renting.books" instanceId="2"/>
16      <target locationId="renting.reader" instanceId="3"/>
17      <user id="1"/>
18    </event>
19    <event action="rent.summary" name="" type="button"
20      startDate="2011-11-27T08:04:00.271"
21      endDate="2011-11-27T08:04:00.271">
22      <source locationId="renting.reader" instanceId="3"/>
23      <target locationId="renting.summary" instanceId="4"/>
24      <user id="1"/>
25    </event>
26  </events>
27</log>

```

Figure 4.3: Example for the events log descriptor

Könyvtári rendszerek riportja (Mon Nov 28 14:56:47 CET 2011)

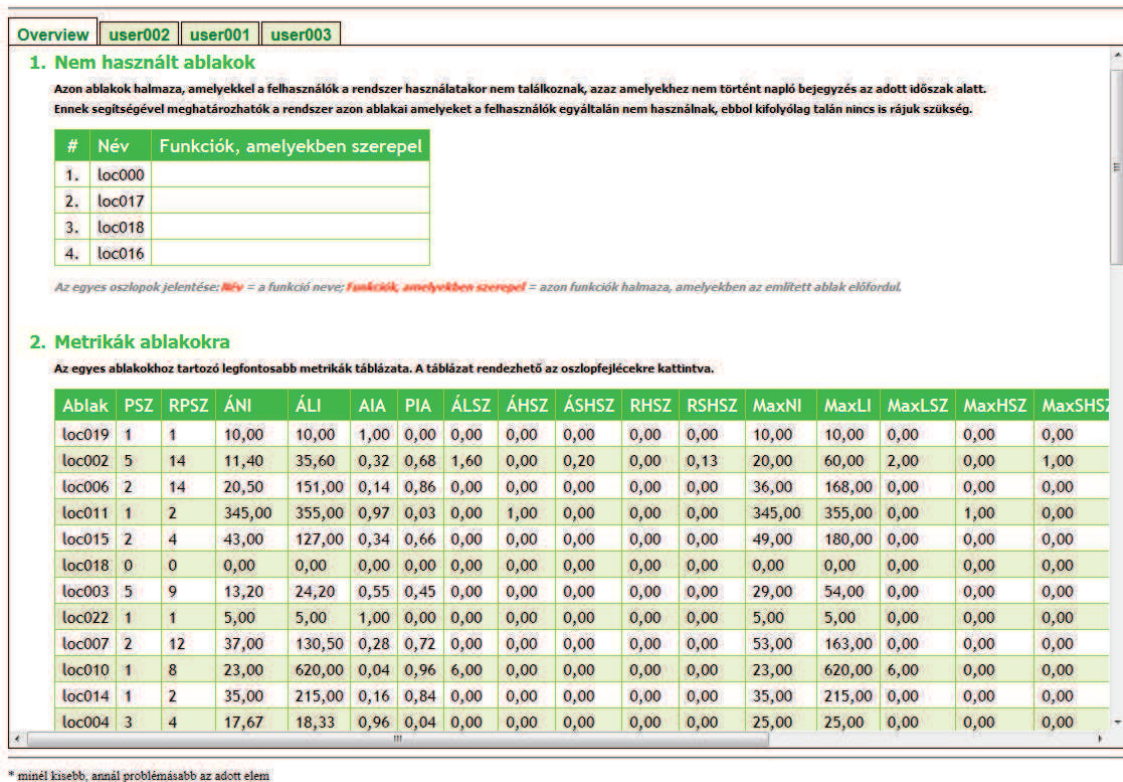


Figure 4.4: Example report (in Hungarian)

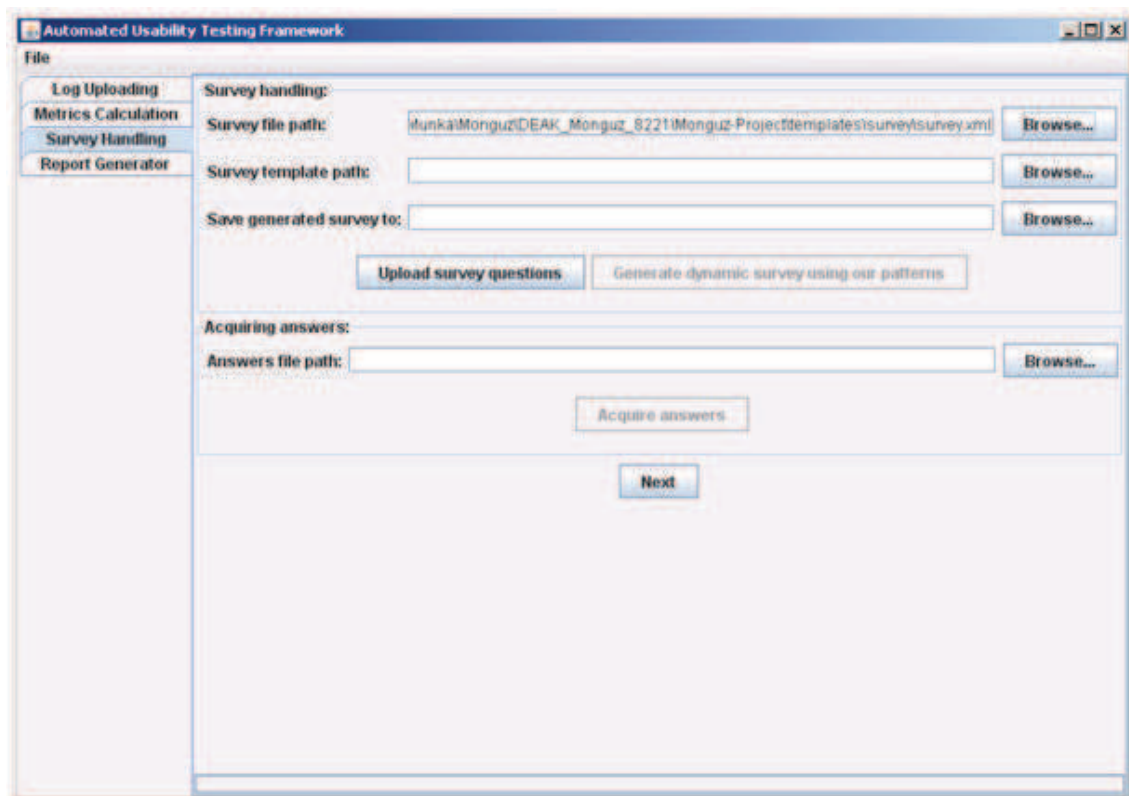


Figure 4.5: User interface of the AUEF

Chapter 5

Case study

5.1 Integration and operation

We have applied UEF in two libraries operating in Hungary. We are allowed to refer the investigated systems anonymously, so we refer them as System1 and System2. We have deployed software upgrades in these libraries to facilitate the collection of usage data, which are stored in the collection systems' database.

The functionality of the software was not affected during the data-gathering phase, the only development was the addition of a logging module that monitored user actions and stored them in the database for later collection.

The data was gathered during the normal daily operation of the participating libraries over a period of two weeks. The study did not affect the nature or workflow of the tasks performed by the librarians. Typical daily activities include circulation-related transactions, such as the loaning of books, placing holds on unavailable items and circulation-related financial transactions (mostly overdue return fees) and cataloging (the recording of bibliographic and item data on items that were newly acquired or not cataloged before).

The study included all members of staff of the participant libraries who use the integrated system, without distinction to experience or position. Most librarians have been using the system for years, while newcomers receive formal or informal training in its use.

5.2 Results

The first interesting result is that there were no unused windows in System2 while System1 has 654. The cause of this phenomenon is that System2 has a specific installation only with the necessary functions but System1 has a full installation. Furthermore, the examination period was a little bit short (two weeks) due to the strict deadlines of the

project.

Metric id	Metric name	Short description
NI	Number of Instances	Number of instances for a location type
RNI	Recursive Number of Instances	Number of instances for a location type plus instances of other windows opened from this location type
NT	Net Time	Net time spent on a location instance
ET	Existence Time	The time elapsed between the opening and closing of a location instance
ATR	Active Time Ratio	For a location instance, it is the ratio of having focus to the time of existence
PTR	Passive Time Ratio	For a location instance, it is the ratio of not having focus to the time of existence
NS	Number of Steps	Number of steps for completing a functionality starting from the given location instance
NE	Number of Errors	Number of error messages appeared from a given location instance
NHU	Number of Help Uses	Number of help uses initiated from a given location instance
RNE	Relative Number of Errors	The ratio of error messages appeared to the number of steps while performing a functionality starting from the given location instance
RNHU	Relative Number of Help Uses	The ratio of help uses to the number of steps while performing a functionality starting from the given location instance
RT	Recursive Time	The time spent over the full functionality accessible from the given location instance (in seconds)

Table 5.1: Some instance level metrics

Patterns

We have examined the top 30 patterns found in each system. The summary of these is shown in Table 5.2.

	System1			System2		
	Min	Max	Avg	Min	Max	Avg
Pattern freq.	11	59	22.03	3	57	14.17
Pattern's length	10	16	10.7	5	16	6.3

Table 5.2: Summary of the patterns found in the examined systems

There is a quite long sequence in System1 with a very high number of occurrences. This could be a hint for the developers to revise the task behind this pattern, and maybe create a simpler window or wizard containing this functionality, as a cohesive unit. The average pattern length is significantly bigger in System1 than in System2. The frequency difference is slightly smaller, but still brings us to the conclusion, that System1 is a little bit harder to handle, it takes many steps to perform a certain task.

Metrics

We have selected a few metrics that could possibly help the evaluation of usability in different systems. Seffah and others [5] introduce more than one hundred metrics for usability. Unfortunately, in the current project we have had to show results quickly and keep the strict and agile deadlines. Therefore, we only had resources to examine and derive metrics from the productivity aspect [5]. This basically includes time and step based metrics, ratios, and a few combined metrics. Some of the calculated instance level metrics are summarized in Table 5.1. A complete collection of the calculated metrics (and other materials) are available on the project's homepage [4].

The metrics are calculated for both of the systems. The summary of the most important ones is shown in Table 5.3.

Questionnaires

The questionnaire setup contains the following questions:

- *How satisfied are you with the integrated system in general? (1-10, 1: very unsatisfied, 10: very satisfied)* From these votes, an average has been calculated and showed in the report.
- *How many windows you generally have open at the same time? (<5, 5-10, 10-15, 15-20, 20<)* If this is a relatively high number, then it shows that some of the windows (locations) should be simplified or merged, or some hidden dependencies between the windows should be fixed.
- *How often do you use the built-in help? (Never, Rarely, Reasonably often, Often)* It can enlighten whether the system is difficult to use or maybe the help is not so useful, and therefore the users do not even try to turn to it.
- *Please summarize your personal experiences with the system. Can you think of any way it can be enhanced? (free text)* It can show any further usability problem with the investigated applications.

The survey has been filled out by 20 users in the case of System1, and by 50 users in the case of System2.

Relation between metrics and questionnaires

There is a question enquiring about the overall satisfaction of the users. Users of System2 gave an average 7.4 grade out of 10, while users of System1 marked their system with an average 6.89 score. Some relation can be observed when comparing these subjective

	System1			System2		
	Max	Avg	Med	Max	Avg	Med
NI	461	28	8	115	22	8
RNI	1753	292	18	291	34	8
NT	6628	169	10	4348	296	19.5
ET	95446	9269	33	37445	3755	7.9

Table 5.3: Most important metrics for the two examined systems. Lower or equal values of the comparison of System1 and System2 are denoted with gray background.

results to the calculated objective metrics. NT-Avg and NT-Med, the metric values of System2 is lower than the ones of System1 (see Table 5.3). Based on the other metrics, we hypothesize that lower metric values for NI, RNI and ET indicate better overall satisfaction of the users. It seems trivial that low NI (Number of Instances) indicate higher usability, because the user has to understand and manage less locations in the application. Low RNI (Recursive Number of Instances) and low ET (Existence Time) values indicate that working with a location is relatively fast, i.e. performing a certain operation, which is represented by a location, is relatively easy.

There are some textual responses given by the users. We have compared these to the calculated metric values. The "Title.OpenDialog" window has an average of half minute net time, but there is a user who has spent six minutes on this window. There is a person, who filled out the questionnaires, complaining about this window, and suggests an improvement between the relation of the input fields, which could improve the search results. This leads to the conclusion that this window could be refined in order to be more efficient and easier to use. Another example is the "Renew" window, having a high average net time, and the maximum net time of thirty minutes. Another person suggests that the librarians should have the opportunity to change the order of the entries in this window. Maybe this is the reason for which they have spent half an hour on this window.

Based on the previous two cases, it seems that in certain cases NI, RNI and ET could indicate the general usability of a software system, while NT could mark tangible usability bugs.

Visualization of user interaction sequences

The UEF administrator can generate a yed graph [6] presenting and visualizing the aggregated user interactions. Nodes represent locations of the investigated applications, while edges between the nodes represent interactions between locations. The background color and the size of a node is determined by how frequently it is used by the users, i.e. large nodes with darker background are used more frequently than others (see Figure 5.1).

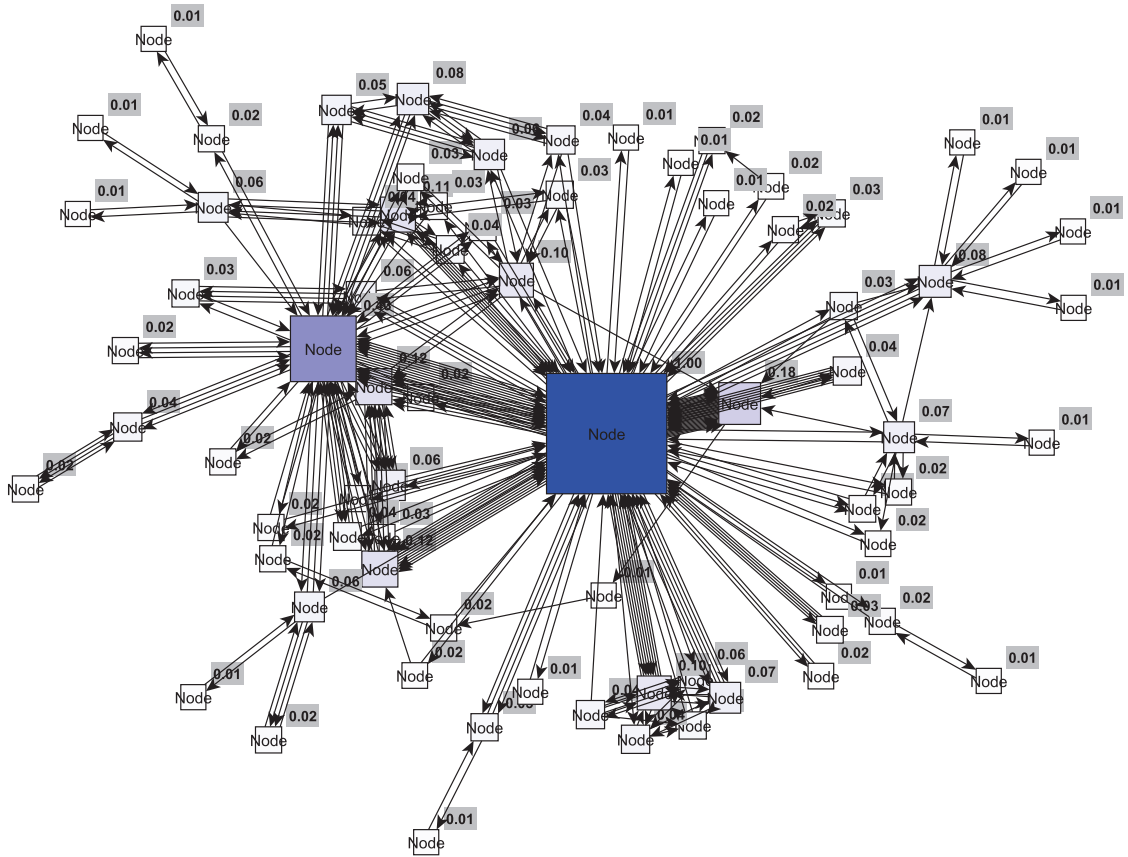


Figure 5.1: Frequency-based visualization of user interaction sequences for System1

5.3 Improvement of the systems

We have made improvements to the ICMS systems based on the results of the UEF report. In the case of System1, some of the results point out exact usability bugs in the system, which were fixed in the following release. Other responses point out the unreliability of the automatic notification system, which is fixed in an unreleased version of the product. The data also pointed out possible usability improvements in certain windows of the circulation module, some of which are improved upon, other improvements are being planned in the next major release of the software.

The main line of development based on this data is the simplification of the circulation workflow, which currently requires the user to go through a large number of steps to achieve daily tasks. Current plans include intelligent skipping of steps based on defaults and a wizard-like interface for certain tasks.

In Figure 5.2 we present the opinions given by the users of one of the ICMS systems we have examined.

Könyvtári rendszerek riportja (Mon Nov 28 14:56:47 CET 2011)

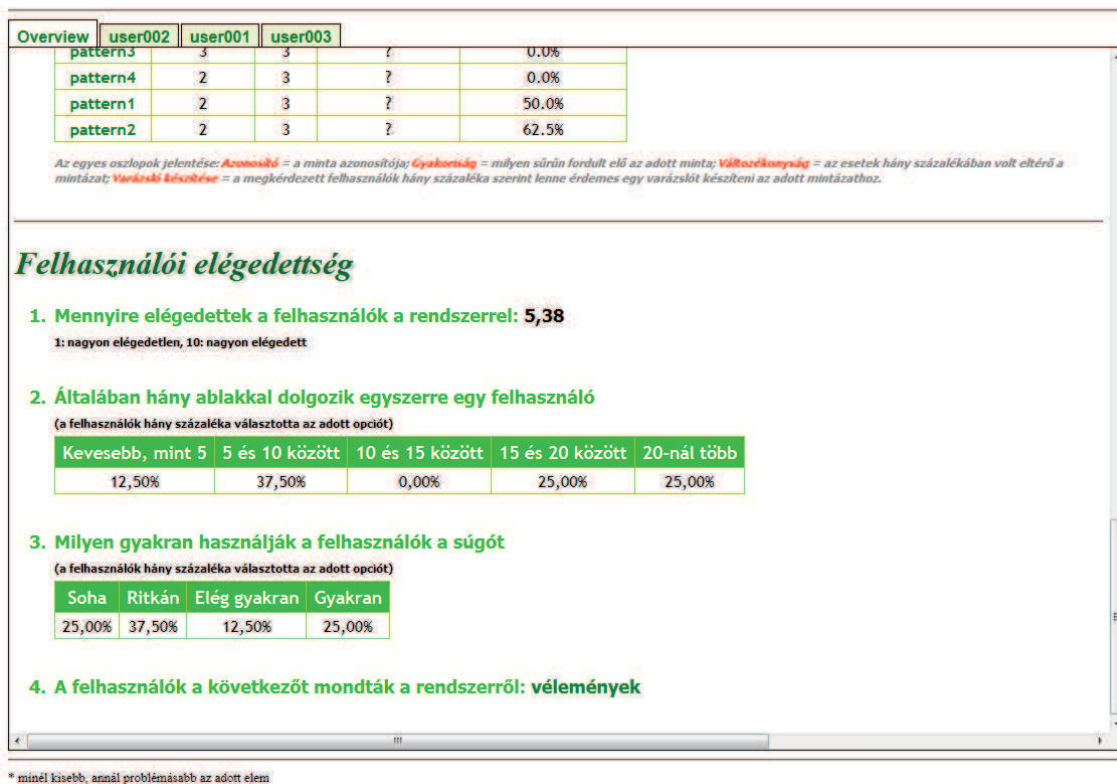


Figure 5.2: Example report with questionnaires (in Hungarian)

Chapter 6

Evaluation

In this section we evaluate UEF through the requirements introduced in Section 2.

- *Evaluation validity*
 - Identification of *red routes* is completed by detecting *Sequences of User Actions*. Furthermore, the frequency-based visualization of user interaction sequences can be also used to identify red routes and important locations.
 - *Subjective* is completed by the application and integration of an open source survey engine. In the case study, users filled the generated surveys and the data was loaded back to the UEF prototype.
 - *Concrete bugs* is completed by the questionnaire and the calculated metrics.
- *Transparency* can be achieved by the data being recorded in the background, so the users are not affected. The framework ensures this requirement by its general XML input formats. The original application is responsible for silent logging and to provide the XML files.
- *Automatic* is mostly completed. UEF only requires one manual step, filling the questionnaires by the users.
- *Domain-independent* is also completed because the UEF's general usability model and interface does not contain any ICMS specific data.
- *Application-independent* is also completed because the UEF's data model and interface does not contain any application specific data. Application specific logging mechanisms are independent from the UEF.
- *Platform-independent* is guaranteed by that the framework is written in Java, and its interfaces are also platform-independent: the input can be given in XML format, while the HTML results can be displayed in any browser.

Chapter 7

Related work

Schretnner et. al [7] shortly introduced UEF together with other projects. However, that paper has not introduced a lot of important details and results about the presented UEF framework and about the presented case studies.

One key requirement of the UEF framework is application-independency (see Section 2.2), i.e. the developed framework has to be generally applicable, for example in the case of desktop applications as well. This way, we cannot apply such a complete and popular solution like Google analytics [8] or other web log analyzers.

Au et. al. [9] describes the aspects of usability testing, including the usability testing of mobile systems. The authors of the paper examined some systems and the results show that it is recommended to use an automated usability testing framework instead of manual usability testing. This helps the developer to test the system more often, even in the early stages of the development, when the changes are cheaper. The paper presents the authors' own automated usability testing framework called HUIA. The framework records the users' actions and then compares them with the expected actions. The comparison shows how differently the users are doing the actions than it is expected.

Ivory et. al. [10] introduce the current techniques of usability testing and then they compare and evaluate the different approaches with each other. The main comparison is the level of automation. The authors suggest making a unified usability testing system, because the current approaches are giving very different results. The paper introduces the concept of usability coverage. The authors are managing the usability testing of the WIMP (Window, Icon, Mouse, Pointer) and the Web applications differently. Their comparison is done by dimensions: capture, analysis, critique. The evaluation shows that the current usability testing techniques use poor automation. In the end, the authors suggest directions to improve the current approaches with automation.

Harty et. al. [11] is about the automation of the usability and accessibility testing of web-based applications. The author describes why is it so hard to do usability and

accessibility testing. After this, the paper discusses about the difficulty of automation. There are a lot of academic attempts to do usability testing, but most of them are not usable in an industrial environment. Another difficulty of the web-based usability automation is that there is a lot of different web browsers and techniques which makes it hard to do the automation.

Comparison. We compare UEF to other existing solutions. The previously cited and several other papers [12, 13, 14, 15, 16] also deal with usability evaluation. However, to the best of our knowledge, there is no such usability evaluation technique available that conforms to our requirements like the presented Usability Evaluation Framework. Usability testing and evaluation is typically executed in a staging or testing environment, while UEF works in the production environment of the investigated system. Furthermore, available techniques usually concentrate on a specific domain, e.g. web log analyzers work only in case of web applications, while UEF is general, it is domain-, platform- and application-independent.

Chapter 8

Conclusions

The contribution of the paper can be summarized as follows. We have defined the requirements of a general usability evaluation framework, and we also defined a general usability model. Based on the requirements and the model, we developed the architecture of the framework. Aside from some technical details the presented architecture can be considered as a reference architecture.

Following the architecture, we also developed a prototype and successfully applied it to evaluate and to improve the usability of two ICMS systems. The ICMS systems are evaluated in real-life working production environments, they are used by the workers of two libraries in Hungary. However, we applied the framework in the field of ICMS, it is platform-, domain- and application-independent. We are currently working on the application of UEF in other domains, i.e. on the usability evaluation of an office software suite in mobile and desktop application contexts. We also plan to integrate further metrics into the framework and to perform further case studies in the future.

Acknowledgement

First of all, we would like to say thanks to our supervisors who have coordinated our work in this project and for their help in making of this paper. We would also like to thank the participation of **Monguz Ltd.**, who gave us input data to test our framework on, and for their willing to negotiate about the correct format of these files we needed.

Bibliography

- [1] P. Grubb and A. A. Takang, *Software Maintenance: Concepts and Practice*. World Scientific Publishing Company, 2003.
- [2] T. K. Landauer, *The Trouble with Computers: Usefulness, Usability, and Productivity*. A Bradford Book, 1995.
- [3] “The userfocus homepage.” [Online]. Available: <http://www.userfocus.co.uk/articles/redroutes.html>
- [4] “UEF homepage.” [Online]. Available: "<http://www.inf.u-szeged.hu/~flajos/usability>"
- [5] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda, “Usability measurement and metrics: A consolidated model,” *Software Quality Control*, vol. 14, no. 2, pp. 159–178, Jun. 2006.
- [6] “The yEd homepage.” [Online]. Available: http://www.yworks.com/en/products_yed_about.html
- [7] L. Schrettnner, L. J. Fülöp, A. Beszédes, A. Kiss, and T. Gyimóthy, “Software Quality Model and Framework with Applications in Industrial Context,” in *Proceedings of 16th European Conference on Software Maintenance and Reengineering (CSMR’12)*, 2012.
- [8] L. Hasan, A. Morris, and S. Proberts, *Using Google Analytics to Evaluate the Usability of E-Commerce Sites*. Springer Berlin / Heidelberg, 2009.
- [9] F. T. W. Au, S. Baker, I. Warren, and G. Dobbie, “Automated usability testing framework,” in *Proceedings of the ninth conference on Australasian user interface*, vol. 76, Darlinghurst, Australia, Australia, 2008, pp. 55–64.
- [10] M. Y. Ivory and M. A. Hearst, “The state of the art in automating usability evaluation of user interfaces,” *ACM Computing Surveys*, vol. 33, pp. 470–516, Dec. 2001.

- [11] J. Harty, “Finding usability bugs with automated tests,” *Communications of the ACM*, vol. 54, pp. 44–49, Feb. 2011.
- [12] M. Runge, *Simulation of Cognitive Processes for automated Usability Testing*. Diploma, Deutsche Telekom Laboratories, Berlin, 2008.
- [13] A. Hosseini-Khayat, T. D. Hellmann, and F. Maurer, “Distributed and Automated Usability Testing of Low-Fidelity Prototypes,” in *Proceedings of the AGILE conference*, 2010, pp. 59–66.
- [14] M. S. Andreasen, H. V. Nielsen, S. O. Schröder, and J. Stage, “What happened to remote usability testing?: an empirical study of three methods,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 2007, pp. 1405–1414.
- [15] P. K. Chilana, A. J. Ko, J. O. Wobbrock, T. Grossman, and G. Fitzmaurice, “Post-deployment usability: a survey of current practices,” in *Proceedings of the 2011 annual conference on Human factors in computing systems*, New York, NY, USA, 2011, pp. 2243–2246.
- [16] S. Propp and P. Forbrig, “ViSE: a virtual smart environment for usability evaluation,” in *Proceedings of the Third international conference on Human-centred software engineering*, Berlin, Heidelberg, 2010, pp. 38–45.