A Semi-Automatic Usability Evaluation Framework

Kornél Muhi[1], Gábor Szőke[1], Lajos Jenő Fülöp[1], Rudolf Ferenc[1], and Ágoston Berger[2]

[1] University of Szeged
Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
{mkornel,kancsuki,flajos,ferenc}@inf.u-szeged.hu
[2] MONGUZ Ltd.
Nemestakács u. 12/A H-6722 Szeged, Hungary
aberger@monguz.hu

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 development of the framework is driven by well-defined requirements. It is built upon a 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.

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, Landauer [10] reports an interesting distribution of software 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. Albeit, it is 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. Namely, a usability lab is a simulation of a real production environment thus some usability bugs could be missed. Another really promising direction is remote usability testing but some of the usability bugs still could be missed.

Therefore, usability testing should be supported by tools that can be applied in production environments 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 arbitrary software applications, i.e. independently from programming language, operating system, and other factors (e.g. desktop, mobile and web applications).

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 arbitrary software application as well. We have two goals with the framework. We want to develop a completely general framework that is widely applicable, and we want to test and improve the usability of two industrial software systems.

The paper is organized as follows. In the next two sections we describe the related work and elicit the requirements for the framework. In Section 4 we introduce the key component of the framework, the general usability model. Next, we show the details of the framework in Section 5. Section 6 shows our results about applying the framework in case of two industrial software systems. Finally, the last two sections present the evaluation of the framework and the conclusions.

2 Related work

In our previous short paper [15] we described and introduced our general software quality model and framework with the underlying principles and methodology. UEF was just briefly introduced as one of the applications of this general softare quality framework, in less than a half page, so that paper did not introduced a lot of important details and results.

There are some very good and popular solutions available like Google Analytics [5] and TrackerBird [17]. These solutions collect data about user behaviour that can be used broadly, e.g. to derive usability problems [7]. Although, these solutions work on specific fields. For example, TrackerBird works on .NET applications while Google Analytics works on web pages.

Au et. al. [2] describes the aspects of usability testing, especially in case of mobile applications. They examined some systems and the results show that it is recommended to use an automated usability testing framework. This helps the developer to test the system more often, even in the early stages of the development, when the changes are cheaper. They also developed and presented an automated usability testing framework.

Ivory et. al. [9] compare and evaluate usability testing approaches. The authors suggest making a unified usability testing system, because the current approaches are giving very different results. The evaluation also shows that the current usability testing techniques use poor automation.

Harty et. al. [6] is about the automation of the usability testing of web-based applications. The authors describe why is it so hard to do usability and accessibility testing. They conclude that there are a lot of academic attempts for usability testing, but most of them are not usable in an industrial environment.

Several other papers [14] [8] [1] [4] [13] also deal with usability evaluation. To sum up, usability testing and evaluation is typically executed in a staging (testing) environment, and constrained to a special application field or domain. For example web log analyzers cannot be employed in case of desktop applications, .NET specific analyzers cannot be applied in case of Java applications, and so on.

3 Requirements

The following requirements are set up based on (i) the investigation of related work and (ii) after several discussions with professionals from the industry and our university.

Support production environments: Usability testing and evaluation are typically executed in a usability lab, where a moderator controls and observes the representative users. Several usability bugs can be detected with this technique but the laboratory circumstances also determine missed usability bugs. Therefore, the usability evaluation shall be performed in real-life *production environments*.

Detection of patterns: The usability evaluation shall reveal typical and frequent sequences of user interactions, i.e. *patterns*. For example, such detected frequent sequences could be optimized and handled with more attention.

Detection of usability bugs: The usability evaluation shall provide tangible *usability bugs*. For example, it can be a frequent and complex user interaction sequence 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: The usability evaluation shall be *transparent* and not affect the daily operational work of the users. 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.

Automatic: The usability evaluation shall be *automatized* as much as possible, especially in users point of view. The rationale behind this requirement is to reduce extra costs generated by unnecessary manual work.

Wide applicability: The usability evaluation shall be as *general* as possible. It should be domain independent, i.e. it should be applied arbitrary in different application domains, e.g. in case of financial applications, ERP applications, office applications and so on. It should be applicable in different kind of operating systems (e.g. Windows, Linux, etc.). Furthermore, it should be applicable in different environments (e.g. in case of desktop, web or mobile applications). Finally, the usability evaluation should be applicable in different kind of programming languages as well (e.g. Java, C++, C#, etc.).

4 Usability model

A well defined data model (shown on Figure 1) is the key to our approach, and the basis of the other components of the UEF framework. The model is developed through several iterations. In each iteration, the model is evaluated and then improved based on the previously shown requirements. In this section, we introduce the final model in detail.

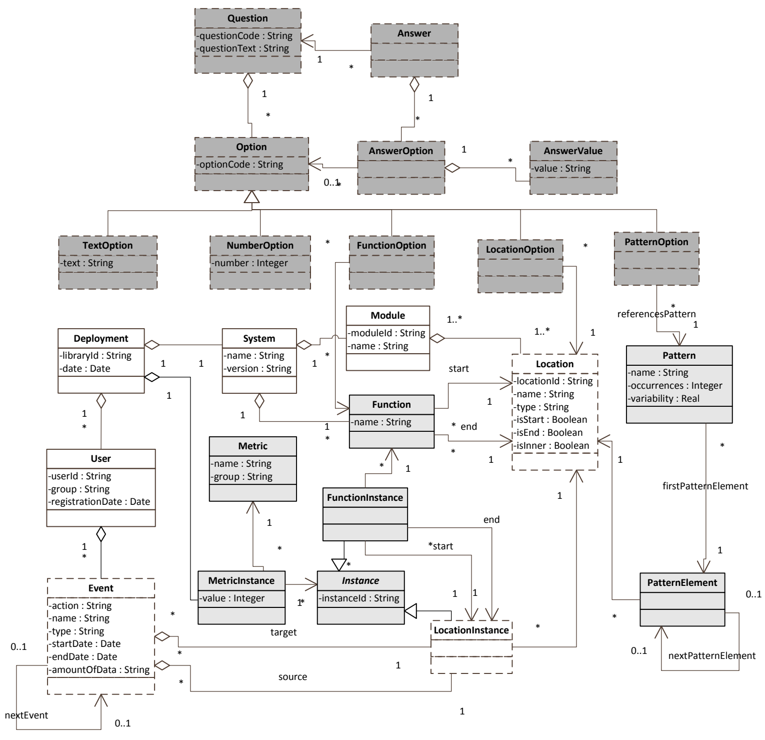

Fig. 1. The usability model

System nodes are denoted by non filled nodes with continuous borders in Figure 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 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 it's name, type and other information. LocationInstance represents a concrete instance of a Location. For example, in several applications it is possible to open multiple 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 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 it's first element, and then a PatternElement refers the next element recursively.

Survey nodes are denoted by filled nodes with dashed borders in Figure 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 *AnswerValue* is introduced to store comments and justifications about the answers. Patterns can also be referenced in the survey (*PatternOption*).

5 Framework

In the followings, we shortly introduce the architecture of the framework that is also based on the requirements (see Figure 2). The arrows and numbers represent the working steps in chronological order.

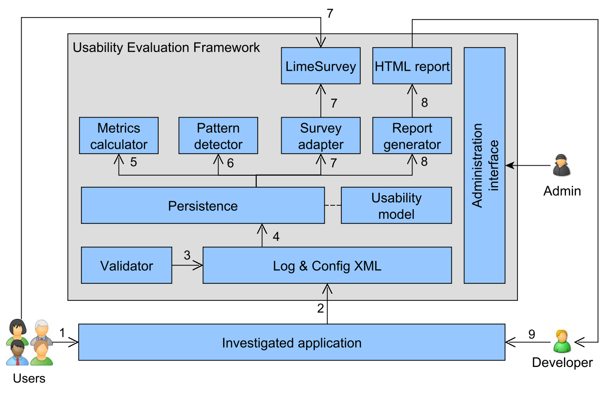

Fig. 2. Architecture of the UEF framework

The first step is to record user interactions that shall be performed by the investigated application. The second step is the generation of the framework input files by the investigated application. These two steps ensure that the framework and its services are completely independent from any kind of technological platform (e.g. operating system, programming language, etc.).

The framework has two kinds of input files. *Config XML* describes meta information about the system, e.g. registered users, available modules and available locations (windows, menu panels, etc.), see Figure 3. Based on these metadata, *Log XML* provides the concrete usage data (see Figure 4). The development of these XML formats is heavily influenced by the usability model.

The validator component checks the logfiles according to predefined syntactic and semantic rules. Then, the logged data are uploaded into the persistence framework for further processing. The schema of the persistence layer is configured with the *usability data model*. The model represents several information related to usability: events performed by the users, locations touched by the users, and so on (see Section 4).

The metrics calculator module can calculate several numeric or textual properties. This module can be extended, calculation of new metrics requires only a new metric calculator plugin to be developed.

```
<deployment libraryId="library1" date="2011-11-30"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="extformat.xsd">
  <system name="Big Library" version="1.0.0"
      deploymentDate="2005-01-01">
    <module id="module01" name="reader"/>
    <module id="module02" name="operator"/>
  </system>
  <locations>
      <location locationId="renting.books" name="Select books to
          rent" type="window" module="operator"/>
      <location locationId="renting.reader" name="Data of the
          reader" type="window" module="operator"/>
      <location locationId="renting.summary" name="Summary about
          the rent" type="window" module="operator"/>
      <location locationId="renting.help" name="Renting books help"
          type="help" module="operator"/>
      <location locationId="renting.fault" name="Renting books
          error" type="error" module="operator"/>
   </locations>
   <functions>
      <function name="Rent a book" startLocationId="renting.books"
          endLocationId="renting.summary"/>
   </functions>
   <users>
      <user id="1" group="administrator"
          registrationDate="2011-11-30"/>
      <user id="2" group="operator" registrationDate="2011-11-30"/>
   </users>
</deployment>
```

Fig. 3. Example for the configuration descriptor

To process the user events stored in the persistence framework, a pattern detection module has been developed based on the very efficient suffix array method [12]. This method has been widely applied and adopted in several fields of computer science, for example it is employed by software clone detection algorithms as well [3]. In case of UEF, all user interaction is stored and ordered in an array, and then, based on the suffix array method, the patterns of user interactions are detected.

Survey adapter module is capable of generating configuration files which can be imported into the LimeSurvey open source survey engine [11]. These surveys can be filled out by the users of the subject systems, and the results can be uploaded back into the persistence system. Questions can be easily added, modified and removed in the framework. These surveys collect subjetive opinions and can contain questions regarding the results of the current data analysis.

```
<log xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="logformat.xsd"
  libraryId="library1" date="2011-11-30">
 <events>
   <event action="rent" name="" type="button"
      startDate="2011-11-27T08:03:47.103"
      endDate="2011-11-27T08:03:47.103">
     <source locationId="main" instanceId="1"/>
     <target locationId="renting.books" instanceId="2"/>
      <user id="1"/>
    </event>
    <event action="rent.reader" name="" type="button"
       startDate="2011-11-27T08:03:52.163"
       endDate="2011-11-27T08:03:52.163">
      <source locationId="renting.books" instanceId="2"/>
      <target locationId="renting.reader" instanceId="3"/>
      <user id="1"/>
    </event>
    <event action="rent.summary" name="" type="button"
       startDate="2011-11-27T08:04:00.271"
       endDate="2011-11-27T08:04:00.271">
      <source locationId="renting.reader" instanceId="3"/>
      <target locationId="renting.summary" instanceId="4"/>
      <user id="1"/>
    </event>
 </events>
</log>
```

Fig. 4. Example for the events log descriptor

Based on the collected information (metrics, patterns and surveys) the UEF framework generates an HTML report (see UEF homepage [18]). Then the report can be employed by the developers to improve the usability of the investigated application.

We have implemented a prototype of the UEF architecture in Java.

6 Proof of concept

Integrated collection management systems (ICMS) manage the business workflow of libraries and cultural institutes. Monguz Ltd. is one of the leading software development companies in Hungary in the field of ICMS. This company wanted 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 UEF. We are allowed to refer the investigated systems anonymously, so we refer them as System1 and System2. Both systems are implemented in Java, but System1 is a desktop while System2 is a web application. The study included all members of staff of the participant libraries who use the integrated system, without distinction to experience or position.

We have developed *specific loggers* inside the investigated applications to collect usage information. Finally, the collected information are exported into the input format of UEF (config and log XML files).

Based on the literature and discussions with professionals of Monguz Ltd. a few metrics have been implemented in UEF to help the evaluation of the two ICMS systems. Seffah and others [16] introduce more than one hundred metrics for usability in different aspects. We have employed and derived metrics from the productivity aspect. Some of the calculated metrics are summarized in Table 1. A complete collection of the calculated metrics (and other materials) are available on the project's homepage [18].

Metric	**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	Time between opening and closing (location instance metric)
ATR	Active Time Ratio	The ratio of having focus to the time of existence (location instance metric)
PTR	Passive Time Ratio	The ratio of not having focus to the time of existence (location instance metric)
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
RT	Recursive Time	The time spent over the full functionality accessible from the given location instance (in seconds)

Table 1. Some of the calculated metrics

Based on several discussions with professionals of Monguz Ltd. a questionnaire has been set up as well. The survey has been filled out by 20 users in the case of System1, and by 50 users in the case of System2.

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. Correlation models cannot be built between the questionnaires and calculated metrics because we evaluated only two software systems. Still, some relation can be observed when comparing

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

these subjective results to the calculated objective metrics. Except of two cases (NT-Avg and NT-Med), the metric values of System2 are lower than the ones of System1 (see Table 2). Based on the other values, we can hypothesize that lower values for NI, RNI and ET indicate better overall satisfaction of the users.

There are some textual responses given by the users concerning concrete locations in the software. We compared these to the calculated metric values and 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.

We have also examined the top 30 patterns (frequent user interaction sequences) found in both system (see Table 3). The average pattern length is significantly bigger in System1 (22 locations) than in System2 (14 locations). Based on this, we hypothesize that pattern length could be an indicator for usability problems as well (besides the metrics).

	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 3. Summary of the patterns found in the examined systems

6.1 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 problems with the automatic notification system, which is also fixed. 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.

7 Evaluation and application guidelines

In this section we evaluate UEF. First, we have to emphasize that the framework does not deal with application specific logging of user interactions. In some point of view it is a drawback because application developers have to extend their application to generate XML input files for the UEF framework. At the same time, it is a really big advantage in an other point of view. This feature guarantees a very important requirement of the framework, the wide applicability. By generating the necessary XML files, the framework provides several useful features: detection of frequent user interactions (i.e. patterns), calculation of metrics, detection of questionnaires based on predefined templates, and so on.

In the followings, we show a detailed evaluation of the framework through the requirements introduced in Section 3.

- *Support production environments* is indirectly fulfilled by the XML-based configuration files. Nowadays, any kind of software application can produce simple XML text files that can be loaded into the framework. In fact, this requirement have to be fulfilled by the investigated application.
- *Detection of patterns* is fulfilled by the adaptation of the suffix array method and algorithm. In the proof of concept, several patterns have been detected and evaluated.
- *Detection of usability bugs* is fulfilled by the adaptation of a questionnaire engine and the automatically calculated metrics. However, manual investigation and evaluation of the questionnaire results and the calculated metrics is needed. Furthermore, we have calculated some basic metrics only, and validate them in a simple proof of concept experiment. Therefore, this requirement has been only fulfilled yet.
- *Transparency* is fulfilled because 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 investigated application is responsible for silent logging and to provide the XML files.
- *Automatic* is mostly completed. UEF only requires one manual step from users: filling the questionnaires.
- *Wide applicability* is also fulfilled. It is domain- and application independent because its general usability model and interfaces do not contain any domain or application specific data, i.e. domain and application specific logging mechanisms are independent from the UEF. Platform-independency 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.

7.1 General application guidelines

Based on the experiencies collected during the proof of concept we have developed a general 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. It seems that specific loggers will have to be implemented for each new case study to produce specific XML files. In some degree it is true if a concrete software application should be investigated specifically with the logging of application specific data. Still, the extension of the corresponding software application would desire just a little extra work based on our current experiencies[3]. Moreover, specific libraries could be developed to collect technology specific data like TrackerBird [17] does it for .NET applications. This way, usability data could be collected easily from several software applications, which are in the same technology domain.

8 Conclusions

The contribution of the paper can be summarized as follows. Based on related works and several discussions with professionals from industry we have defined the requirements of a general usability evaluation framework. Next, we developed the Usability Evaluation Framework (UEF) conforming to the defined requirements. The provided framework architecture can be considered as a reference architecture for future (e.g. industrial) implementations as well.

We successfully applied UEF to evaluate and to improve the usability of two systems. They are evaluated in real-life production environments, at two libraries in Hungary. During the evaluation, the framework has been extended with newly implemented metrics and newly configured questionnaires based on the discussions with the industry partner. The developed metric calculator plugins and the configured questionnaires could be used by new potential partners

[3] The discussed proof of concept has proved that it is easy to develop such XML generator components.

and application developers in the future. Moreover, we have successfully demonstrated that the UEF framework can be extended easily with new metrics and questionnaires to satisfy the needs of an industrial partner.

The UEF framework is designed to be platform-, domain- and application-independent. Contrary, the evaluated two systems are a little bit homogeneous, i.e. they come from the same ICMS domain, both are written in Java while one run in a web browser and the other run as a standalone Windows desktop application. Therefore, we are currently working on the application of UEF in other contexts, i.e. the usability evaluation of an office software suite written in C++ and runs both on mobile devices and as a standalone Windows desktop application. We also plan to integrate further metrics and questionnaires into the framework, and to perform new case studies in the future.

Acknowledgement This research was supported by the Hungarian national grants GOP-1.1.2-07/1-2008-0007 and GOP-1.1.1-11-2011-0006.

References

1. Andreasen, M.S., Nielsen, H.V., Schrøder, S.O., Stage, J.: What happened to remote usability testing?: an empirical study of three methods. In: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 1405–1414. New York, NY, USA (2007)
2. Au, F.T.W., Baker, S., Warren, I., Dobbie, G.: Automated usability testing framework. In: Proceedings of the ninth conference on Australasian user interface. vol. 76, pp. 55–64. Darlinghurst, Australia, Australia (2008)
3. Basit, H.A., Jarzabek, S.: Efficient token based clone detection with flexible tokenization. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 513–516. ESEC-FSE '07, ACM, New York, NY, USA (2007)
4. Chilana, P.K., Ko, A.J., Wobbrock, J.O., Grossman, T., Fitzmaurice, G.: Post-deployment usability: a survey of current practices. In: Proceedings of the 2011 annual conference on Human factors in computing systems. pp. 2243–2246. New York, NY, USA (2011)
5. Homepage of Google Analytics [7 May 2013], `http://www.google.com/analytics/`
6. Harty, J.: Finding usability bugs with automated tests. Communications of the ACM 54, 44–49 (Feb 2011)
7. Hasan, L., Morris, A., Probets, S.: Using Google Analytics to Evaluate the Usability of E-Commerce Sites. Springer Berlin / Heidelberg (2009)
8. Hosseini-Khayat, A., Hellmann, T.D., Maurer, F.: Distributed and Automated Usability Testing of Low-Fidelity Prototypes. In: Proceedings of the AGILE conference. pp. 59–66 (2010)
9. Ivory, M.Y., Hearst, M.A.: The state of the art in automating usability evaluation of user interfaces. ACM Computing Surveys 33, 470–516 (Dec 2001)
10. Landauer, T.K.: The Trouble with Computers: Usefulness, Usability, and Productivity. A Bradford Book (1995)
11. LimeSurvey - Open Source Survey Application [7 May 2013], `http://www.limesurvey.org`

12. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. In: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. pp. 319–327. SODA '90, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1990)
13. Propp, S., Forbrig, P.: ViSE: a virtual smart environment for usability evaluation. In: Proceedings of the Third international conference on Human-centred software engineering. pp. 38–45. Berlin, Heidelberg (2010)
14. Runge, M.: Simulation of Cognitive Processes for automated Usability Testing. Diploma, Deutche Telekom Laboratories, Berlin (2008)
15. Schrettner, L., Fülöp, L.J., Beszédes, A., Kiss, A., Gyimóthy, T.: Software Quality Model and Framework with Applications in Industrial Context. In: Proceedings of 16th European Conference on Software Maintenance and Reengineering (CSMR'12) (2012)
16. Seffah, A., Donyaee, M., Kline, R.B., Padda, H.K.: Usability measurement and metrics: A consolidated model. Software Quality Control 14(2), 159–178 (Jun 2006)
17. Homepage of TrackerBird [7 May 2013], `http://www.trackerbird.com/`
18. Homepage of UEF [7 May 2013], `http://www.inf.u-szeged.hu/~flajos/usability`