Automated usability measurement of arbitrary desktop application with eyetracking

Pavol Fabo
Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava, Slovakia
Email: pavol.fabo@fmph.uniba.sk

Roamn Ďurikovič
Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava, Slovakia
Email: durikovic@fmph.uniba.sk

***Abstract*—Nowadays in software development process more attention is paid to the final usability of the product. To achieve such usability we use various methods from user centered design up to the usability evaluation methods, requiring much attention from usability experts. The presense of these experts are needed both during capturing and analysing usability data, which eventually costs too much. We propose a tool for automated data capturing during user tests as well as a captured data analysis in order to evaluate the recorded interaction and guide the attention of software developers. Furthermore we provide a simple statistics of user tests as well as a means to browse recorded data with the interaction context extended with eye tracking data.**

***Keywords*-Usability measurement, Automated interaction measurement, Interface evaluation**

I. Introduction

In the history of software development, there are many cases where inadequate usability of software have lead to a disaster. Thus more and more attention has been paid to the user centered design and overall usability of the software products.

There are many ways to ensure the usability of sofware. First of all a design guidelines and/or user interface patterns are used, then new software development models like iterative or spiral models have been devised, thus overcomming the drawbacks of traditional waterfall software development model. Finally the software's usability is evaluated using tests with the users, usually in usability laboratories. In such laboratory a special environment is set up to enable usability expert to record the interaction of the tester. The recorded data are than evaluated by usability experts in order to analyse several key parts of the interaction. Since this process uses humans to record and evaluate it is extremely time consuming. To target the excessive time consumation of this standard usability evaluation approach a various other methods of interface evaluation and usability testing have been created [1], [2], [3].

Another approach to overcome the time consuming drawback of traditional usability test is the automation. Automated usability measurements and testing are very promising area of usability research. Since the usability testing process consists of three stages, i.e. capturing, analysis and critique, the automation might be employed in each and every of these stages.

Automated interaction data capturing is used today in a standard usability laboratories using video cameras capturing the user and even the computer screen. But such data are video based thus are very hard to analyse automatically. On the other hand a various interaction logging systems have been used instead to capture the interaction. Automation in the analysis and critique stages are not that straightforward. Here a various models and metrics have to be used in order to make an usability conclusion about the interaction.

II. Usabilv application a tool for automated usability evaluation

The automation of the usability evaluation process has been done several times earlier. The automation has been employed in large variety of evaluation methods from testing, inspection, inquiry, analytical modeling up to the simulation methods [4].

Automated usability testing methods usually capture events and timestamps of the events and consequently analysing this data using various metrics or pattern-matching approaches [4]. Such methods unfortunatelly does not involve the user, just the events generated by the user.

Inspection methods are based on an examination of the user interface and its conformance to a set of guidelines. Even here an automation has been done, mainly in the analysis process with respect to the graphical user interface elements, widgets and layout, thus eventually ommiting the analysis of the user's interaction itself.

Inquiry methods are motivated with capturing of the user's subjective impressions about various aspocts of an user interface [4]. The biggest drawback of the inquiry methods is the lack of automation in the analysis and critique processes. This is mainly due to the lack of formalism, since the data are generated by the users themselves.

Analytical modeling usability evaluation methods use some kind of model, either the model of the user interface, or the model of the user himself, or both in order to help the evaluator to predict the usability of the interaction [4]. The well known GOMS model, which focuses on the

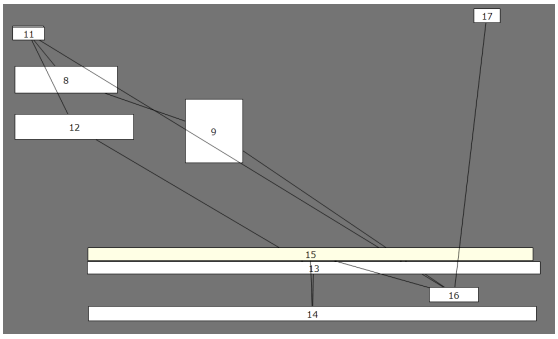

Figure 1. Model of the interaction in Microsoft Paint application. Actual modelled interaction is for opening and saving an image using jpeg compression. Model consist of eight click events (white rectangular boxes) and one text event (yellow rectangular box) labelled with respective ID numbers.

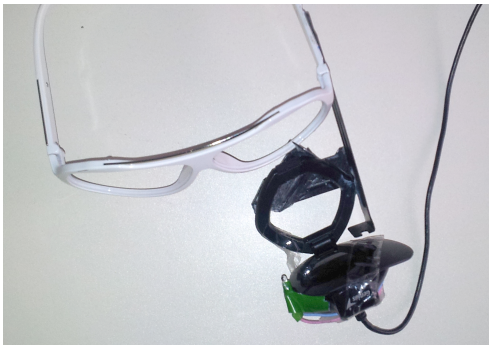

Figure 2. Prototype of low cost eyetracker consisting of an old glasses and a near infrared sensitive web camera.

user performace, is also an analytical modeling evaluation method [5].

The last group of usability evaluation methods is the simulation methods group. The simulation methods are based on the simulations of the user's interaction, which is done using the models of the user interface and/or the user himself [4]. Automated interaction data capturing is done by simply generating the data for the simulation by specified algorithms like the genetic ones.

A. Interaction model definition

Since our intention was to evaluate any desktop application we have to omit an usability models bounded to a specific software or interface. On the other hand we use a model of the interaction, created by defining the sequence of events to be performed by the user. By drawing nodes and ordering nodes into a sequence, the interaction model is created very intuitivelly.

We distinguish five different type of nodes for model definition. Four type of nodes cover mouse events i.e. mouse click, mouse double click, mouse up and mouse down, and the fifth node type covers text events i.e. key press event. An example of interaction model is in Figure 1

The interaction model definition is extended with an approximate time the user should complete the interaction during the test.

B. System event tracker

The interaction data capturing is done by system event tracker. It uses an UserActivityMonitor library [6], which enables global event handling. We distinguish five system events specifically mouse up, mouse down, mouse click, mouse double click and key press. Every event, along with the time stamp and in case of mouse events with the mouse position and mouse state, is stored in database. Furthermore when an event, which is significantly effecting the look of the interface, especially the mouse clicks and doubleclicks occurs, a screenshot of actual screen is grabbed and stored along with the event id to the database. This is done asynchronously, thus the image capturing, compression and storing into the database does not affect the interaction smoothness.

C. Eye tracker

The system event data itself is able to define the user's interaction very well. Unfortunatelly such data does not contain information about the user itself. To capture this type of data we use a low cost head mounted eyetracker shown in Figure 2. Eyetracker's data is used to compute a point of user's gaze on the computer's screen as described in [7].

Our eyetracker consists of a standalone web camera sensitive in a near infrared spectrum, which is extended with an infrared illumination LED mounted directly to the camera. Using the infrared illumination coupled with the sunglass's lens used as a pseudo camera filter, we have been able to achieve proper lighting of the user's eye, thus the process of the detection of the user's gaze is computationally very simple. For simplicity we consider the position of the user's head is static.

In the detection process we use median filter with square kernel of size of five pixels. Using threshold operation and contour detection we are able to precisely localize the user's pupil. By computing image moments defined by Equation 1. and described in [8] we are able to compute the coordinates of the centroid of the user's pupil using Equation 2.

$$m_{p,q} = \sum_{i=1}^{n} I(x,y) x^p y^p \tag{1}$$

$$\begin{aligned} x_{centroid} &= \frac{m_{10}}{m_{00}} \\ y_{centroid} &= \frac{m_{01}}{m_{00}} \end{aligned} \tag{2}$$

Using calibration we are able to map pupil's center point to the actual screen coordinates, forming the user's gaze point on the screen as described in [8], [7]. Pupil's center is

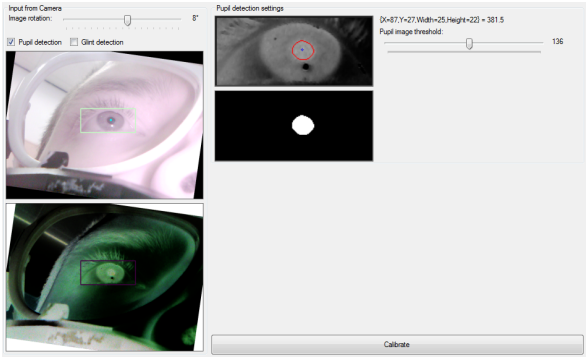

Figure 3. Realtime pupil detection using standalone computer vision algorithms. Top left web camera image shows a definition of range of interest for pupil detection. Bottom right web camera image shows a thresholded image. By finding contour and image moments a pupil center is defined in top right image.

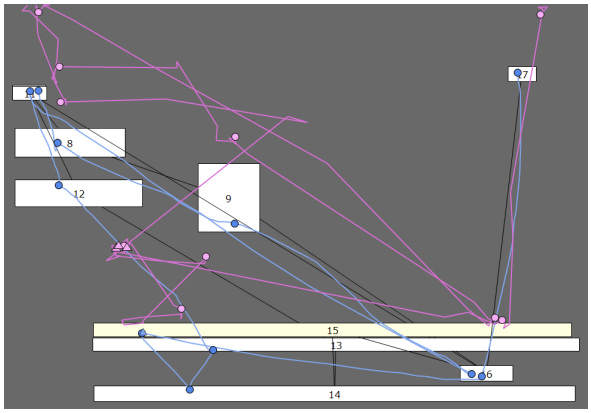

Figure 4. Captured system event data (blue lines) and user eyetracked data (violet lines) put into the context of the interaction model. A discrapancy between eyetracked data and system event is clearly visible, showing not optimal parts of the interface.

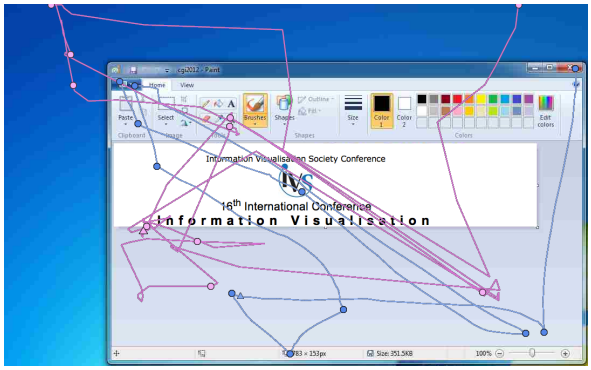

Figure 5. Representing correspondence between system events and eyetracker data as a red dashed line. The interaction context in form of a screenshot, which has been stored in a database, is shown as well.

computed as a floating point coordinates, thus the resulting screen coordinate point is computed with subpixel accuracy. The computer vision algorithms are implemented using a EmguCV library [9].

In Figure 3 the results of computer vision algorithms and pupil center detection are shown. Notice the quality of an input image, which enables far simpler and less time consuming pupil's centroid detection.

D. Recorded data presentation

An essential part of any usability testings is a possibility to browse the interaction data and virtually play the entire user's interaction. Such feature is very needed in order to make it possible to further examine the interaction data.

In our application, since we store every available data in a database, we are able to play back the user's interaction. In Figure 4 a stored system event data as well as user eyetracker data are shown in blue and violet colors. Putting the recorded interaction in the context of the defined interaction model we are able to visually see any interaction errors. Blue and violet circles in Figure 4 represent the mouse click events and actual gaze position during the mouse click. Further a violed triangles represent that the gaze position when a key has been pressed during the interaction.

Note the positional difference between the system event data shown as blue lines, and the eyetracker data, shown as violet lines, which is caused by not having a static head position during eyetracker data capturing. We consider such misposition to be a great feature from the interaction exporation point of view. The misposition visually separate eyetracker data from system event data thus promoting the visual exploration.

The interaction records shown if Figure 4 are very good to assess the overall interaction.

In order to enhance the visual analysis we have denoted the actual correspondence of the eyetracker data and the system events data using a red dashed line as shown in Figure 5. What's more we are able to playback whole interaction. The position is visually represented as a big rectangle both in system events data (blue rectangle) and eyetracked data (violet rectangle). Furthermore, since we have stored a interface screenshot after each system event, we are able to show the context of the interaction i.e. the actual user interface as shown in Figure 5.

E. Simple interaction analysis

Our first step into the usability evaluation was to detect the most evident problems in the interaction. We have considered various performance metrics, as defined in [10], and found out that three of them are very straightforward to determine from the captured data.

Firstly, we consider the actual time of the user's interaction. During the interaction model definition we have defined the approximate time it should take the user to successfully complete the interaction. During the user test we keep track

of time by inserting a timestamp information to the database along with the both system event data and eyetracker data. After the test we are able to find the time of the interaction by simple database query. This time is compared with the time defined in the interaction model. If the difference is big enough, we have used difference of two seconds, the interaction is tagged with a time difference problem, denoted as a ”T” letter in the ”Problems” column in Figure 6.

Table name	Problems	Success
test01_noEye	T	Minor
test02		Minor
test03	T	Success
test04	T	Success
test05	CT	Failure
test06	CT	Major

Figure 6. Users interaction records enhanced with simple interaction analysis tags.

Secondly, the number of system events, especially mouse clicks, is another very important usability indicator. Since the interaction model consists of actual events, that the user should perform, we are able to detect if the user actually performed these events. Such comparison is not that straightforward since we need to take into account the text nodes in the interaction model. Sometimes the text node need to be click inside, other times it gains focus during the interaction. Thus we count the minimum and maximum click count of the interaction model. These are then compared with the interaction the user has performed during the tests. If the user's click count is not inside the interval defined by minimum and maximum interaction model click counts, the interaction is tagged with a click count problem, denoted with ”C” letter in the ”Problems” column. It needs to be said that such approach is not the optimal one, but for the purposes of simple analysis it should be sufficient. We perform more detailed analysis, which is described in Section II-F.

Finally, the last metric we use for simple analysis is distance travelled. The interaction model has defined the position as well as the order of the nodes. From the position of nodes we are able to compute the distance that should user travel in order to reach the next node. We compute an interval defined by the minimum and the maximum distance the user might travel. Minimum distance, is the distance of the closest nodes points, the maximum distance is the distance of the two farthest points. In the database, the information about user's mouse movement are stored, thus the computation of the travelled distance by the user is very straightforward as shown in Equation 3.

$$d = \sum_{i=0}^{n-1} (P_{i+1} - P_i) \tag{3}$$

Eventually if the users distance travelled is not in the defined interval, the entire interaction is tagged with a distanced problem, denoted by ”D” letter in the ”Problems” column.

Except of these metrics, we use also a task success metric, which is recorded at the end of the user interaction by user's subjective assessment, thus providing the tests with subjective data, which is also very useful.

These three metrics along with the subjective task success metric form our simple analysis, which is used as a first look on the interaction.

F. Enhanced interaction analysis

Using a simple analysis we have provided a quick first look on the quality of the user interactions during the tests. Using an enhanced interaction analysis we aim to validate the recorded interaction to the interaction model, especially considering the order and the position of the system events.

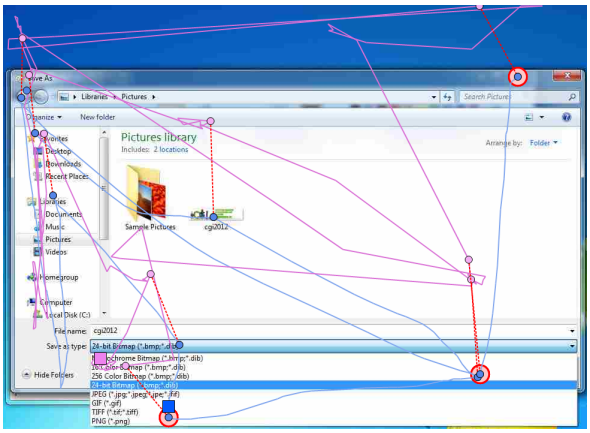

Figure 7. Showing a problem in the interaction. The user has not performed the right interaction by missing the click node and eventually missing all successive nodes. All missed targets are shown as red circles.

The validation of mouse events is straightforward, on the other hand text nodes in the interaction models has to be targeted specifically. The text node is very special since a user might use as many mouse events in the text node as he wants. Thus the mouse click, doubleclick, up and down events inside the text node are not considered. As soon as the mouse events are outside the text node they are again taken into the account. Using this validation process we are able to find mispositioned events, which eventually affects the interaction. Such interaction problem is shown in Figure 7.

G. Test statistics

To provide a global view on the usability test a interaction statistics are available. We have implemented task success statistics, which are computed per test, as shown in Figure 8.

Further more a most valuable metric, time on task, is also available in stats. In Figure 9 the blue column denote a mean

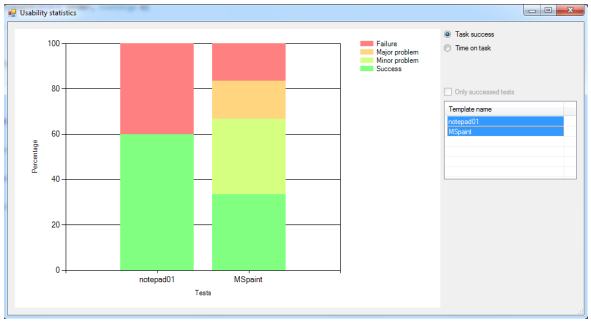

Figure 8. Showing the percentual representation of task success for individual tests. Test called "notepad01" in the left column of the graph has 60% success rate shown as green color and 40% failure rate shown as red color. Test called "MSpaint" in the right column of the graph has about 33% success rate shown as green color. Around another 33% of tests have minor problems, shown as lime color bar. Orange color denotes major problems during the tests, whereas red color denotes test failure.

of time on tasks through all interaction records in a test. This is extended with a standard deviation shown as red error bars. The orange column denote the time of the interaction as defined in the interaction model.

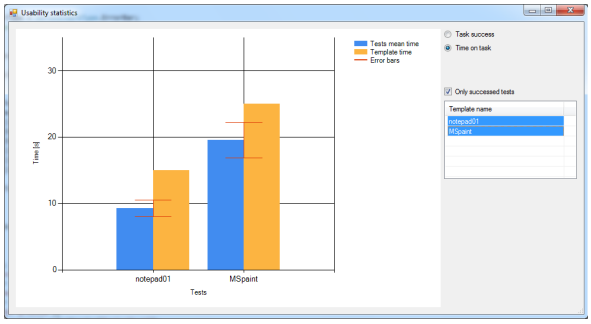

Figure 9. Statistics comparing the mean time of the task (blue column) of all test to the interaction model defined time (orange column). The graph is enhanced with a standard deviation error bars (red color).

III. Conclusion and Future work

We have presented an novel approach for automated usability evaluation. We have automated the data capturing and shown four metrics, i.e. time success, travelled distance, click count and time on task, which are used for simple analysis. Furthermore the validation of system events order and position form an enhanced analysis. By making available evaluation statistics, we make it possible to evaluate the overall interaction usability.

The biggest advantage of our method is its versatility. It can be used for virtually any desktop application. The only mandatory thing needed for the usage of our method is the interaction model definition.

We are aware of several drawback of our method. For instance text nodes needs to be distinguished, since there is an interaction difference between the textarea nodes and textbox nodes. Furthermore, we need to consider changing the focus of the user interface elements, by pressing either the tab or alt+tab key. Our approach assume that the user will not customize the environment during the test. Simple displacement of user interface elements will lead to the total test failure, since the positions of interface model nodes are defined absolutely. Finally, we need to take into account, that there are several ways of the interaction, leading to the task success.

These and any other problems are target of our future research in the field of the automated usability evaluation.

IV. Acknowledgement

This research was partially supported by a VEGA 1/0898/12 2012-2014 a Scientific grant from Ministry of Education of Slovak Republic and Slovak Academy of Science and by Comenius University Grant.

References

[1] W. Albert, T. Tullis, and D. Tedesco, *Beyond the Usability Lab: Conducting Large-scale Online User Experience Studies*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[2] J. Nielsen, *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. [Online]. Available: http://portal.acm.org/citation.cfm?id=529793

[3] J. S. Dumas and J. C. Redish, *A Practical Guide to Usability Testing*, 1st ed. Exeter, UK, UK: Intellect Books, 1999.

[4] M. Y. Ivory and M. A. Hearst, "The state of the art in automated usability evaluation of user," Berkeley, CA, USA, Tech. Rep., 2000.

[5] B. E. John and D. E. Kieras, "The goms family of user interface analysis techniques: comparison and contrast," *ACM Trans. Comput.-Hum. Interact.*, vol. 3, pp. 320–351, December 1996. [Online]. Available: http://doi.acm.org/10.1145/235833.236054

[6] G. Mamaladze, "Processing global mouse and keyboard hooks in c#," 2004. [Online]. Available: http://www.codeproject.com/KB/cs/globalhook.aspx

[7] A. T. Duchowski, *Eye Tracking Methodology: Theory and Practice*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[8] G. Bradski and A. Kaehler, *Learning opencv, 1st edition*, 1st ed. O'Reilly Media, Inc., 2008.

[9] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

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