Temporal Eye-Tracking Data: Evolution of Debugging Strategies with Multiple Representations

Roman Bednarik and Markku Tukiainen
Department of Computer Science and Statistics
University of Joensuu, Finland*

Abstract

The challenges in empirical eye-tracking studies of usability or complex problem solving include 1) how to effectively analyze the eye-tracking data, and 2) how to interpret and relate the resulting measures to the user cognitive processing. We conducted a reanalysis of eye-tracking data from a recent study that involved programmers of two experience groups debugging a program with the help of multiple representations. The proportional fixation time on each area of interest (AOI), frequency of visual attention switches between the areas, and the type of switch were investigated during five consequential phases of ten minutes of debugging. We increased the granularity of the focus on the user processing several times, allowing us to construct a better picture of the process. In addition, plotting the areas of interest in time supported a visual analysis and comparison with the quantitative data.

We found repetitive patterns of visual attention that were associated with less experience in programming and lower performance. We also discovered that at the beginning of the process programmers made use of both the code and visualization while frequently switching between them. At a later stage of debugging, more experienced programmers began to increasingly integrate also the output of the program and employed a high-frequency of visual attention switching to coordinate the three representations.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces - Evaluation/methodology—Input devices and strategies;

1 Introduction

The technological problems of eye-tracking systems are being continually resolved, making the technique more usable and easier to apply. Modern eye-tracking systems are easy to operate, make no interference with participants, and are claimed to reliably capture a large proportion of population. At the moment, eye-tracking is considered a common tool of an HCI analyst.

Nonetheless, the issues related to data analysis and interpretation prevent HCI researchers and practitioners from utilizing eye-tracking further. As the most challenging, [Jacob and Karn 2003] list two methodological problems: labor-intensive data extraction and difficulties in their interpretation. Commercial systems are often supplied with a recording and analysis software that reduces the labor associated with the manual fixation extraction. While this automation can facilitate the analysis for simple and short tasks, the studies of complex processing with modern interactive systems present a new challenge to eye-tracking researchers.

*e-mail: firstname.surname@cs.joensuu.fi

ETRA 2008, Savannah, Georgia, March 26–28, 2008.
© 2008 ACM 978-1-59593-982-1/08/0003 $5.00

The other problem is the interpretation of the data in evaluating user strategies. In particular, the retrospective relation of the eye-tracking measures to the underlying processing is hard. Typically, the analysis starts from delimiting the scene into the areas of interest, continues through aggregating the data, and ends with computing the measures with respect to the areas. Most of the previous steps can be performed using a software analysis tool. The final step of the analysis, that is the linking of the measures to the phenomena in question, however, is a hard task left to the HCI analyst.

Dealing with large amounts of complex behavioral data is common in many domains. In studies that employ eye-tracking, one or more groups of participants receive a treatment while their ocular behavior is recorded; the researcher then compares the respective aggregated eye-tracking measures between the treatments or groups to confirm or reject the hypothesis. While this approach can be functional with short tasks in range of tens of seconds, such as in the usability studies, in the eye-tracking studies of complex problem-solving the task the participants perform is severalfold longer and also more complex.

The analysis of such eye-tracking data, interpretation of the measures, and relation to the underlying processes cannot be approached as has been done with the short tasks in past [Bednarik and Tukiainen 2006]. Complex tasks are composed of hierarchies of simpler tasks and stages, and therefore the conventional approaches to the analysis, conducted under typical experimental settings, do not accurately uncover the underlying processes. Instead, using the conventional approaches, an mixture of dynamic processes is described using a single eye-tracking measure.

There clearly is a need for advancing the methods of eye-tracking analysis in more complex domains. In this paper we discuss the analysis issues in the context of studies that present several adjacent representations of a computer program and investigate the problem-solving strategies. The present analysis expands on the automatic analysis methods in studies of usability evaluation and seeks to improve them to allow for a better interpretation of the results. We employ two alternative approaches to the analysis of visual strategies. First, to tackle the problem of too coarse analysis, we increase the granularity by segmenting the whole stream of visual attention data into shorter sequences. We also use plots of areas of interest as they were attended during the process, which allows us to compare the information that can be gained using these two approaches.

1.1 Related Work

Linking eye-tracking data to underlying cognitive processes have become the primary challenge in retrospective eye-tracking studies. Practical and some methodological aspects of eye-tracking in usability research have been previously discussed in the work of [Goldberg and Wichansky 2003] and [Jacob and Karn 2003]. For instance, in the eye-tracking studies of usability [Jacob and Karn 2003] argue that the challenge of linking eye-tracking data to the underlying processes has been *"probably the single most significant barrier to the greater inclusion of eye tracking"*. There are several reasons underlying the problem, including, for example, the large amounts of data, complexity of the tasks being studied, or ineffective analysis methods.

Group	N	Age	Program. experience	Bugs found
Expert	8	25.88 (3.94)	108.00 (22.22)	2.75 (1.04)
Novice	6	26.17 (6.08)	42.00 (14.70)	1.50 (0.55)
t (12)		.11 (p=.92)	6.29 (p<.001)	2.67 (p=.02)

Table 1: *Number of participants in each group, their age (SD), programming experience in months (SD), and bugs found (SD) max=4. Differences in groups on independent sample t-test (p-value).*

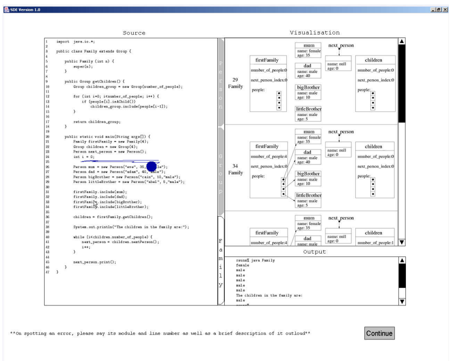
To combat the labor associated with the manual processing of eye-tracking recordings in the usability studies, previous eye-tracking research proposed numerous eye-tracking measures that allow for automation of the evaluation process. Thus, the large quantities of raw eye-tracking data can be significantly reduced to make the analysis of the data more efficient. However, how to interpret the results and relate them to the underlying processing or to the usability aspects are tasks yet not very well understood.

Analysis of underlying processing based on visual attention data becomes popular also in the studies of programming. Relating the eye-tracking data to the underlying processes in programming is not, however, an easy task. This is due to the fact that programming is a complex domain involving many cognitive processes, knowledge and skills that need to be applied to understand multiple and often hidden or implicit components and dependencies. Previous visual attention studies in programming focused, for example, on source code reading [Crosby and Stelovsky 1990], on use and coordination of multiple representations [Romero et al. 2003] or on the effects a visualization of a program has on the visual attention patterns. The studies of visual attention in programming often make use of the hypothesis testing paradigm. In these studies, often, the resulting long-term eye-tracking measures between two groups are compared. [Bednarik and Tukiainen 2006] however argued that *"the comprehension process ... cannot be effectively examined by studying long-term averages [of eye-tracking measures]"*.

The attempts to automatize the analysis and to gain understanding of how to relate the resulting measures to the processing have often employed short and artificial tasks, and long-term eye-tracking measures. It is an open question into which extent the quantitative approaches can be assumed to expose the relation of eye-tracking data and measures to the complex processes involved during problem-solving. This gap motivates us to expand the knowledge available about the methods to analyze and interpret eye-tracking data to the more complex domains. We examine the temporal changes in the eye-tracking measures during a complex problem-solving task with multiple program representations.

2 Case Study: Visual Attention During Debugging with Multiple Representations

We investigated visual attention during debugging. The research settings were similar to those of eye-tracking usability studies: participants were provided with a familiarization task, were not restricted in the interaction with the environment, and the tasks were resembling real world. In this report we make use of a part of the experimental data that has been collected in a replication study reported in [Bednarik and Tukiainen 2007]. We extend the analysis by segmenting the data sets into a series of shorter intervals and thereby we include the temporal aspect into the analysis. We also present illustrative fine-level views on the visual attention by plotting the areas of interest as attended in time. Research settings, materials, and procedures were kept identical to those of the original study [Romero et al. 2003], and we describe them briefly next.



```
import  java.io.*;

public class Family extends Group {

    public Family (int n) {
        super(n);
    }

    public Group getChildren() {
        Group children_group = new Group(number_of_people);

        for (int i=0; i<number_of_people; i++) {
            if (people[i].isAChild())
                children_group.include(people[i-1]);
        }

        return children_group;
    }

    public static void main(String args[]) {
        Family firstFamily = new Family(4);
        Group children = new Group(4);
        Person next_person = new Person();
        int i = 0;

        Person mum = new Person("eve", 35, "female");
        Person dad = new Person("adam", 40, "male");
        Person bigBrother = new Person("cain", 10,"male");
        Person littleBrother = new Person("abel", 5,"male");

        firstFamily.include(mum);
        firstFamily.include(dad);
        firstFamily.include(bigBrother);
        firstFamily.include(littleBrother);

        children = firstFamily.getChildren();

        System.out.println("The children in the family are:");

        while (i<children.number_of_people) {
            next_person = children.nextPerson();
            i++;
        }

        next_person.print();
    }
}
```

```
rsunx% java Family
female
male
male
male
male
male
The children in the family are:
male
```

On spotting an error, please say its module and line number as well as a brief description of it outloud

Figure 1: *A screenshot of IDE employed in the study, with 1 second of gaze overlaid.*

2.1 Method, Participants, and Materials

Figure 1 presents a screenshot of the integrated development environment (IDE) employed in the study. The IDE presented the source code of a program (left in Figure 1) that contained four non-syntactic errors. The other available representations were a visualization of the program (top right in Figure 1) and the output of the program (bottom right in Figure 1).

We collected 14 quality eye-tracking recordings (Tobii 1750) out of 18 participants. Two groups were formed, a highly experienced (hereafter experts, N = 8) and less experienced (hereafter novices, N = 6). Table 1 presents an overview of the two groups, showing significant differences in experience and performance. Three Java program were used that consisted of tens of lines of code and several classes. For each program, the participants – after reading the specifications of the desired behavior of the program – were given ten minutes to debug the program. Participants did not know how many bugs in total there were in the code, and the IDE did not allow modifying of the source code.

To deal with the complexity of the data, the whole ten minute sessions were divided into five two-minute segments. *Proportional fixation time* (PFT) for each of the three areas was computed as a ratio of the fixation time on an area to the overall fixation time on all areas. *Number of switches between areas per minute* was computed as the sum of all changes per minute in visual attention focus between any of the three main areas.

2.2 Results

Figure 2 presents the distribution of PFT. For the subsequent analyses of PFT, only data from code and output were used, because the PFT for code and visualization were almost perfectly negatively correlated (r (5) = -.971, p = .006). A 5 x 2 x 2 (segment, area, experience) ANOVA revealed the main effects of segment (F(4,48) = 4.53, p = .003, η^2 = .274), area (F(1,12) = 765.14, p < .001, η^2 = .985), and experience (F(1,12) = 6.36, p =.027, η^2 = .346). While there was no significant interaction between segment and experience (F(4,48) = .242, ns), the analysis revealed a significant interaction between segment and area (F(4,48) = 3.57, p =.012, η^2 = .229). Other two and three-way interactions were not significant.

The main effect of segment was analyzed using multiple comparisons with Bonferroni adjustment. This showed that while PFT's during last two phases were almost equal, the PFT during the first two minutes was significantly different than during the second segment (p = .037) and nearly significantly different than during fourth segment (p = .053). The difference between second and third segment was not significant. The results indicate that the experts relied more on the source code of the program than the less experienced programmers during all segments. Output of the program became more important than visualization at later phases of the debugging

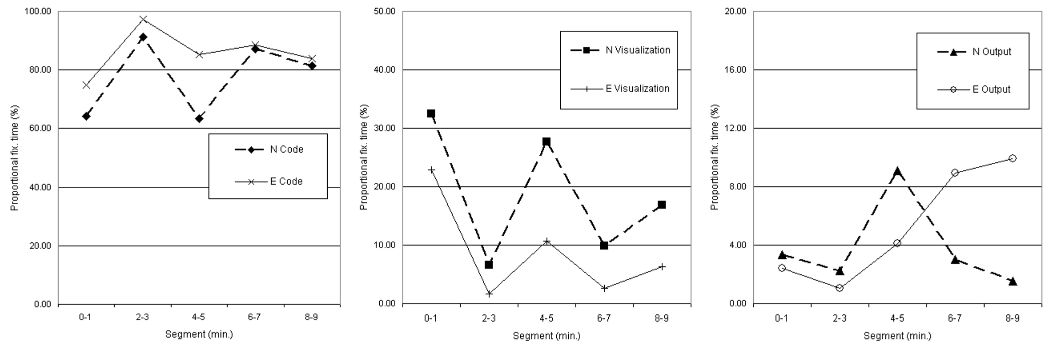


Figure 2: *Plots of proportional fixation times on code (left), visualization (center), and output (right) for novice (N) and expert (E) groups during five phases of debugging. Not in the same scale.*

Segm.	Novices		Experts	
(min.)	sw/m	SD	sw/m	SD
0-1	8.00	4.57	8.63	7.95
2-3	2.42	1.66	1.19	1.19
4-5	8.03	4.30	6.75	5.88
6-7	5.58	3.40	7.50	7.51
8-9	6.42	4.47	9.18	6.18

Table 2: *Switches per minute between any of the three AOIs during the five segments of debugging.*

	Novices			Experts		
Segm.	Code - Visual.	Code - Output	Visual. - Output	Code - Visual.	Code - Output	Visual. - Output
0-1	5.83	0.58	1.58	6.31	1.00	1.31
2-3	1.92	0.42	0.08	1.06	0.06	0.06
4-5	2.25	3.17	1.83	3.75	1.88	1.13
6-7	3.67	1.00	0.92	1.75	**5.00**	0.75
8-9	**5.22**	0.43	0.77	2.06	**5.39**	1.73

Table 3: *Switches per minute for each of the three main types of switches during the five segments of debugging. Note, Visual. = Visualization.*

of experts, while novices tended to attend the visualization.

The frequency of switching was analyzed using a 5 x 2 (segment, experience) ANOVA. Table 2 presents the *overall* number of switches per minute. The analysis revealed the main effect of segment (F(4,48) = 3.99, p = .007, η^2 = .250). Experience had no effect on the number of switches (F(1,12) = 0.11, p =.745, η^2 = .009) and there was no interaction between experience and segment (F(4,48) = 0.477, p =.753, η^2 = .038). Adjusted multiple pairwise comparisons showed that first and second, second and third, and second and fifth segments differed significantly (p = .024, p = .014, p = .005, respectively). Other pairwise differences were not significant.

We examined the relationship between the type of switch, the segment of debugging, and the expertise. Three types of switch were possible: between code and visualization (or back), between code and output (or back), and between visualization and output (or back). Table 3 provides an overview of a breakdown of the switching frequency from Table 2 into the tree types of switches. A 5 x 3 x 2 (segment, switch type, experience) ANOVA revealed significant main effects of segment (F(4,48) = 3.75, p = .01, η^2 = .238) and type of switch (F(2,24) = 9.23, $p < .001$, η^2 = .435) on the switching frequency. The effect of experience was not significant (F(1,12) = 0.18, ns). The two-way interactions of a segment and experience, and of the type of switch and experience were not significant. The interaction of segment and type of switch (F(8,96) = 4.75, $p < .001$, η^2 = .284) and the three-way segment, type of switch and experience interaction (F(8,96) = 4.82, $p < .001$, η^2 = .286) were significant.

Pairwise comparisons showed that the main effect of switch type was due to the switch between code and visualization being significantly more frequent than the switch type between visualization and output (p = .001). The two other comparisons were not significant, although the switch between code and visualization was notably more frequent than the switch between code and output (p = .19) and the switch between code and output was more frequent than the switch between visualization and output (p = .18).

Segmentation allowed us to analyze the fixation patterns in time and how well the eye-tracking measures correspond with the fine-level strategies. The data presented here show the saw-like patterns of visual attention, in particular for the novice group. The PFT on each AOI was not constant during the process, but oscillated between 64% to up to 97%. The overall frequency of switches and the PFT on the code were negatively correlated (r(5) = -.814, p = .093); therefore when the source code was used the most, programmers tended to not to switch to different AOIS. The actual visual attention strategies were, however, far more complicated than that.

The quantitative data indicate that programmers attended to the visualization the most at the beginning. During the second phase they concentrated on the source code, while decreasing the coordination activity. In the middle of the debugging, novice programmers again paid more attention to visualization and but also to the output, and switched more frequently between code and output than in the previous phase. Experts also began to attend to the output and to switch their visual attention between the three representations. From the fourth phase, the experts continued having frequent switches between the output and the program code. The plots of novices' PFT and switching behavior continued in a saw-like pattern. At the final stage of debugging, experts coordinated the representations with the highest frequency.

Figure 3 displays visual attention strategies of a novice (left) and an expert programmer (right), confirming the quantitative data presented above. The novice programmer did not attend very much to the output of the program, however, attempted to do so. The novice participants also did not switch between the AOIs often. The expert programmers, on the other hand, attended more to the output and coordinated the representations with more frequent switches either toward the end or at later phases.
The analysis above suggests that experts spend more efforts integrating the information from multiple representations. In particular, the visual attention strategies they exhibit suggest that expertise promotes relating the code to the output. To examine whether those who did well (regardless of experience) were also integrating different information sources more, we correlated the eye-tracking patterns with debugging performance. Table 4 presents the correlations of the PFT spent on the output with the number of bugs found. The analysis discovered a strong positive and statistically

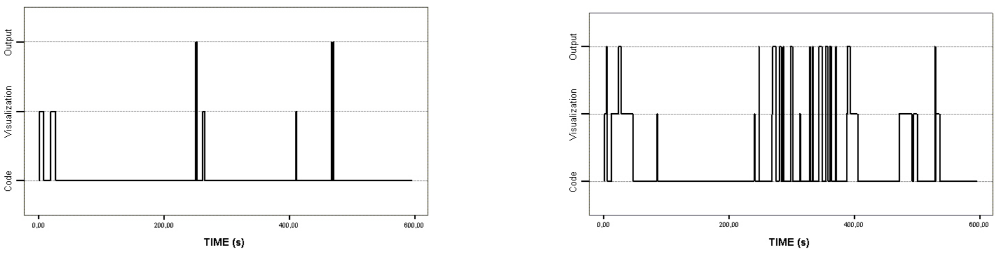


Figure 3: *A plot of the areas of interest as attended during the debugging, displayed left is a novice, right is an expert.*

Segm.	1	2	3	4	5
PFT	-0.07	-0.05	-0.07	0.282	0.73
p	*.81*	*.87*	*.82*	*.33*	*.003***
Switching	-0.20	0.09	0.08	0.17	0.60
p	*.49*	*.76*	*.80*	*.56*	*.02**

Table 4: *Correlation coefficients and associated two-tailed probabilities (N=14) of the PFT spent on the output with performance and of the switching frequency and performance during the five segments of debugging.*

significant correlation in the last phase. Therefore, more successful debuggers employed a strategy that promoted more use of the output in the final stage of debugging. Other correlations of the performance and the times spent on code or visualization were not statistically significant. Table 4 also presents the correlations of the overall frequency of switches (between any AOI) with the number of bugs found. Similarly as with the previous result, the only statistically significant correlation was found during the final phase of debugging. The finding indicates that more successful programmers, regardless of their expertise, increase the representation coordination activities in the last stages of debugging.

In summary, during the initial phases of debugging, we did not find any significant correlations between the visual attention patterns and performance. However, in the later phases, better performance was accompanied by the increased visual coordination activities; in particular, more attention to the output of the program was characteristic to programmers who found more bugs.

3 General Discussion and Conclusions

The analysis and interpretation of eye-tracking data in a rich and dynamic context present a serious challenge. In this study we segmented the data sets into shorter sections, to achieve a finer level of detail about the underlying cognitive activities. The visual attention strategies were analyzed both using the conventional quantitative methods and also by plotting the data points against time and performing a visual analysis.

Our results show that eye-movement patterns during debugging develop in time. Except for the experts' increasing use of output, we however did not find other prevailing trends in the visual attention patterns and representation use, as it has been shown for comprehension tasks [Bednarik and Tukiainen 2006]. Instead, we observed a saw-like pattern of use and segments of frequent switching between the AOI's. More experienced programmers change their strategies during debugging and focus their attention to the output of a program at later stages of the process. The exact moment when they engage in the increased coordination activity, however, differs individually. The results related to switching frequency show that for most of the time it was not sensitive to expertise. Toward the end of debugging more experienced programmers gradually exhibited higher frequency of switching between all three AOI's. The findings indicate that the temporal aspects of eye-tracking data need to be considered as they provide valuable insights about visual attention during lengthly complex tasks.

There are, however, limitations in the automatic methods to analyze the temporal aspects of gaze patterns. In particular, arranging data sets into groups smooths away the individual differences. In [Crosby and Stelovsky 1990] the two most similar scanning patterns while reading an algorithm belonged to subjects from opposite experience groups. Also our study, the individual differences sometimes seemed to predominate over a stereotypical group behavior and caused the variability within a group. These variances impaired the quantitative approaches to variance analysis in their assumptions of homogeneity.

To study individual behavior and strategies, boundaries based on subtasks and events could be determined as references to the behavioral units rather than fixed intervals. For example, one class of such delimitations can be related to a user changing a strategy, e.g. when a (hypothetical) bug has been found. It can be then possible to examine how users modify their strategy on these boundaries. Analysis of that type, however, requires better tools to efficiently analyze variable-length data segments.

In summary, our study shows that segmentation of eye-tracking data in general seems promising, but need to be carried out carefully. We split the data sets according to the pre-defined intervals into shorter segments of equal duration, one of many potential approaches to segmentation. Both the PFT and switching frequency showed sensitivity to the effect of segment. The fine-level plots of visual attention strategies corroborate the findings, and they also highlight the weaknesses of the quantitative methods. The combination of the two analysis methods allowed us to construct a more descriptive picture of the visual attention strategies.

References

BEDNARIK, R., AND TUKIAINEN, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. In *Proc. of ETRA '06*, ACM Press, New York, NY, USA, 125–132.

BEDNARIK, R., AND TUKIAINEN, M. 2007. Validating the restricted focus viewer: A study using eye-movement tracking. *Behavior Research Methods 39*, 2, 274–282.

CROSBY, M. E., AND STELOVSKY, J. 1990. How Do We Read Algorithms? A Case Study. *IEEE Computer 23*, 1, 24–35.

GOLDBERG, J., AND WICHANSKY, A. 2003. Eye Tracking in Usability Evaluation: A Practitioner's Guide. In *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, Elsevier Science, J. Hyönä, R. Radach, and H. Deubel, Eds., pp. 493–516.

JACOB, R. J. K., AND KARN, K. S. 2003. Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises (Section Commentary). In *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, Elsevier Science, J. Hyönä, R. Radach, and H. Deubel, Eds., pp. 573–605.

ROMERO, P., DU BOULAY, B., LUTZ, R., AND COX, R. 2003. The effects of graphical and textual visualisations in multi-representational debugging environments. In *In Proc. of IEEE HCC '03*, IEEE CS, Washington, DC, USA.