Analysing and Interpreting Quantitative Eye-Tracking Data in Studies of Programming: Phases of Debugging with Multiple Representations

Roman Bednarik and Markku Tukiainen

Department of Computer Science and Statistics,
University of Joensuu, PO Box 111, Joensuu, Finland
`firstname.surname@cs.joensuu.fi`

Abstract. While eye-tracking systems become gradually improved and easier to apply, the methodological challenges of how to analyze, interpret and relate the eye-tracking data to user processing remain. Studies of programming behavior are not an exception. We have conducted a reanalysis of eye-tracking data from a previous study that involved programmers of two experience groups debugging a program with the help of graphical representation. Proportional fixation time on each representation, frequency of visual attention switches between the representations and type of switch were investigated in relation to five consequential phases of ten minutes of debugging. Therefore, we have increased the granularity of focus on debugging process.
We found some repetitive patterns of visual strategies that were associated with less experienced programmers finding fewer errors. We also discovered that at the beginning of the process programmers make use of both the code- and graphical representations while frequently switching between them. During the process, more experienced programmers began to integrate also the output of the program and finish the debugging with frequent switching between the three representations. We discuss benefits and limitations of this approach to analyzing and interpreting the quantitative eye-tracking data. As part of future research we propose to investigate the symmetries of representation switching behavior.

1 Introduction

While the technological problems of eye-tracking systems are being continually resolved, granting the increasing usability of the technique, the methodological issues prevent the technology from yet spreading wider. Most challenging – apart from the somewhat remaining technical problems – [1] list two methodological problems with eye-tracking: labor-intensive data extraction and difficulties in their interpretation. Modern eye-tracking systems are easy to operate, make no interference with participants, and can capture up to 90% of population. Commercially available eye-tracking systems are often supplied with recording and analysis software, attempting to reduce the manual data extraction in fixation identification. While this automation can facilitate the analysis for simple tasks, studies of complex processing with interactive systems present a new challenge to eye-tracking researchers.

In many cases the dynamics of the scenes being presented to participants – such as modern computer programming interfaces – makes it hard to link the eye-tracking data to the stimuli. Often, the only solution to this problem is to manually annotate the video-recordings frame-by-frame. Clearly, this approach, besides being uneffective, may bring about several unwanted outcomes. For example, the manual extraction of the fixations might pose a threat to the accuracy of the data analysis. To simplify the process and make it more efficient, automatic quantitative methods of extracting and analysing eye-tracking data have been employed in many studies.

A problem related to automatic eye-tracking data extraction in evaluating interfaces is that of relating the extracted data to the underlying human processing. Typically, the analysis process starts from selecting the eye-tracking measures, continues through delimiting the scene into the areas of interest and aggregating the measures with respect to the areas, to linking the observations to the phenomena in question. In multimedia interfaces that present information both in text and graphics – often by using animation – the analysis phase of an eye-tracking experiment might become the most daunting task of the whole research. To preserve the experimental validity of the eye-tracking experiment, the researcher might not wish to impose constrains on the participants' behavior or employ artificial tasks and environments. Balancing the user freedom at the costs of the constraints of interaction imposed by the study settings can further increase the complexity of the analysis; these factors include, for example, the presentation of several linked representations of the content in adjacent windows, the freedom of users to select when and what representations they want to see, or the possibility of the system to present dialogues such as questions. As the complexity of the interaction increases, however, the link between the eye-tracking data and underlying processing becomes harder to study.

Many studies in programming make use of the hypothesis testing framework [2]. In studies that employ eye-tracking, one or more groups of participants receive a treatment while their ocular behavior is recorded; researcher then compares the respective aggregate 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 of [3] and [4], in eye-tracking studies of learning or problem-solving the task participants perform is severalfold longer and, arguably 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; complex tasks are composed by hierarchies of several simpler tasks and stages, and therefore the conventional methods do not accurately uncover the detailed processes. Instead, using the conventional approaches, an indistinguishable mixture of processes is described using a single eye-tracking measure.

In other words, an averaged measurement is supposed to inform on a complex and long comprehension process.

There clearly is a need for advancing the methodological aspects of eye-tracking research to complex domains. In this paper we discuss one of the challenges for eye-tracking in the studies of program development interfaces, in particular in the studies that present several concurrent representations to investigate problem-solving strategies. The contribution presented here expands on experiences with eye-tracking in studies with multimedia displays evaluation and focuses on methodological issues of automatic eye-tracking data analysis and interpretation.

We begin the discussion by reviewing previous work that considered these aspects in evaluations of computer displays. We then present a case eye-tracking study of debugging strategies with a static program visualization. By segmenting the whole stream of visual attention data into shorter sequences, we increase the granularity of focus on debugging process and we tackle the problem of too coarse analysis. Finally, we also discuss the methodological issues and inherent limitations of this approach to automatic eye-tracking data analysis, as applied in studies of visual attention in programming with multimedia displays.

2 Previous Work

Linking eye-tracking data to underlying cognitive processes have become the primary challenge in eye-tracking studies. For instance, in the eye-tracking studies of usability, [1] argue that this challenge has been *"probably the single most significant barrier to the greater inclusion of eye tracking"*. There are several methodological reasons contributing to the challenge, such as the dynamic nature of modern computer interfaces, the volumes of eye-tracking data, human factors and the concerns of the researchers to retain high validity of the studies.

One domain where eye-tracking has successfully been applied is, undoubtedly, reading research [5]. In studies of reading, eye-tracking made it possible to gain understanding about how visual attention is deployed, how text is processed, or what processes are responsible for guiding the eye during reading. Several models of eye-movement control during reading have been proposed, such as the E-Z reader [6], based on the information obtained from eye-tracking.

In other domains with not as controllable research situation as in reading, such as in studies of usability or driving behavior, the methodological problems of relating the eye-tracking data to the processing have been probably the single most important challenge and barrier to progress. Practical and some methodological aspects of eye-tracking in usability research and studies have been previously discussed in the inspiring works of [7] and [1]. [3] proposed a set of 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 efficient. However, how to interpret the data and relate them to the underlying processing or to the usability aspects are tasks yet not very well understood.

The experimental interface employed in [3] was artificially made and the task given to the participant consisted of only searching for targets with focus on the speed. The interaction with modern computer interfaces, such as with multimedia learning systems, can hardly be considered as simple searching for targets without more elaborate goals. Some studies, such as the web-page usability evaluation of [4], recognize this problem and employ a more realistic task and context. Finding no significant effects of task and page interaction on eye-movement measures, the authors argue for establishing benchmark measures and investigating the relation between underlying processing and eye-movement patterns. They also suggest that visibility of a target might influence the patterns and that more tasks shall be studied to uncover the factors affecting the usability and therefore eye-movement patterns.

Analysis of behavior based on visual attention data becomes popular also in studies of programming. During programming, the programmer has to build an understanding of what the program does and how does it do it to be able to debug and modify the program. Previous studies in programming that employed visual attention tracking focused, for example, on how programmers read the source code [8], how programmers make use of and coordinate multiple representations [9] or on the effects a graphical visualization of a program has on the visual attention patterns of novice programmers [10].

To investigate the link between the underlying processing and overt visual attention patterns eye-tracking researchers often make use of the hypothesis testing framework. For example, [10] presented twelve participants with a program comprehension task and two environments, and compared the resulting visual attention patterns. In another program comprehension study, [11] employed a between-subject design to study differences between expert and novice programmers in comprehension. Similarly as in previous studies, they compared the resulting long-term eye-tracking measures between two groups. [12] however argued that *"the comprehension process ... cannot be effectively examined by studying long-term averages [of eye-tracking measures]"*. In this paper we further investigate this problem of studying the complex mental processes using automatic methods to eye-tracking analysis by examining the temporal changes in eye-tracking measures during a complex problem-solving task with multiple available representations.

In summary, little has been done in past to investigate the link between the eye-tracking data and underlying processing in natural and dynamic computer environments. Researchers have often been left with manual extraction and annotation of the fixation data. Attempts to automatize the analysis of eye-tracking data and to gain understanding of how to relate the resulting measures to the tasks have not arrived to a coherent understanding of the link. Similarly, previous research into the role of visual attention during programming seems to confirm the challenges. This gap motivates our efforts to expand the knowledge available about the connection.

3 Case Study

To prepare the context of the methodological discussion, we first introduce an investigation in which eye-tracking has been applied. We then extend the analysis of eye-tracking data by segmenting the process into a series of several phases and we include the temporal aspect into the analysis.

In the study we investigated the visual strategies of programmers during debugging. The research settings were similar to those of usability evaluations: participants were provided with a familiarization task, were not restricted in the way they wanted to interact with the environment, and the tasks resembled those that the participants would be engaged in under natural conditions.

Integrated development environments (IDE) used by computer programmers often present different views on the program or project in several windows. Programmers normally need to coordinate these representations [13] in order to build a coherent mental model of the program. The representations usually include the source code of the program, output of the program, a visualization of some aspect of the execution or source code, or specifications of the program.

Thus, the goals of the studies related to the cognitive aspects of programming with multiple representations are: 1) to investigate how programmers visually coordinate the representations, 2) whether and how the role of the representations is changing as programmer builds the mental model, and 3) what are the effects of experience on the visual patterns during these programming activities. Ultimately, the answers to these questions can help to better understand the processes involved in learning and problem-solving with multi-representational visualizations.

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 components and dependencies. In fields where eye-tracking has been previously applied, such as in usability studies, eye-tracking researchers often took a quantitative approach to analyze the eye-tracking data. [4], for instance, collected total fixation duration, number of fixations, average fixation duration, and spatial density and investigated their relation to the relative usability differences between different Web-pages.

Similarly as in the usability studies, we have also approached the analysis of the eye-tracking data in a quantitative way. For example, average fixation duration has been suggested in previous research as related to the difficulties with extracting and processing information from a display [3]. Number of fixations, another eye-tracking measure previously employed in studies of programming [14], shall reflect the relative importance of an area of a display. The most essential difference between our studies and the studies of usability, however, is that during learning, problem-solving, or program comprehension the users engage in a variety of complex and relatively lengthy sub-tasks, rather than completing relatively simple short tasks such as search for a target. It is an open question into which extent the quantitative approach to eye-tracking data can be assumed to expose the relations of eye-tracking data and measures to the complexity of the processes involved during problem-solving.

3.1 Case study: Visual attention during debugging with static environment

As a part of a replication study that investigated the effects of the Focus Window Technique (FWT) on the visual strategies during debugging, we have compared the eye-movement patterns of less experienced and highly experienced users. An alternative method to eye-tracking, the FWT was designed to reduce the technical problems with eye-tracking. Whole FWT screen is blurred except for a small section. The study compared the visual strategies of programmers working under the FWT mode to the strategies with unrestricted environment. Research settings of the study, materials, and procedures were kept identical to those of the original study [13], and can also be found from e.g. [14]. The investigation did show an effect of the blurring condition on the behavior of users, and the results have been reported elsewhere [15].

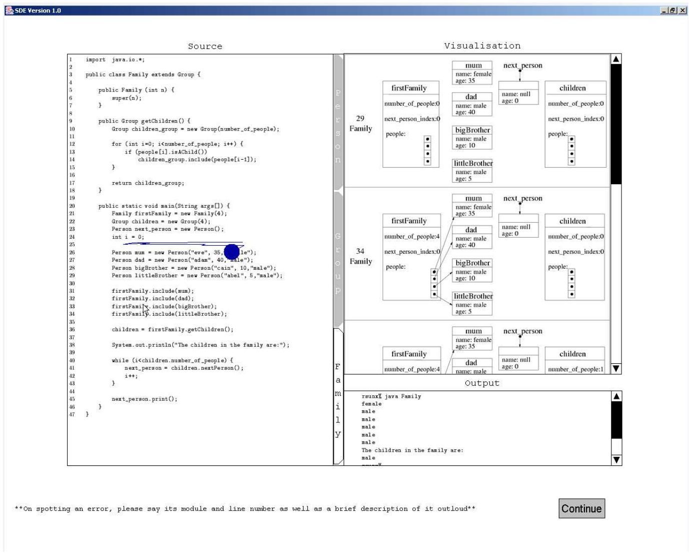
Method Figure 1 presents a screenshot of the IDE during the non-restrictive condition. The environment presented the source code of a Java program (left pane in Figure 1) that contained four non-syntactic errors. Participants – after reading specifications of the desired behavior of the program – were given ten minutes to debug the program. Eighteen participants have taken part in the study. Missing, corrupt or incomplete data were removed from the sample, leaving fourteen quality eye-tracking recordings. Two groups of users, highly experienced (hereafter experts) (N = 8) and less experienced (hereafter novices) (N = 6), were formed from the remaining data. Table 1 presents an overview of the two groups, showing significant differences in experience and performance in terms of bugs found.

Participants were not allowed to modify the source code. Additional representations provided by the IDE were a visualization of the program (shown top right in Figure 1) and the current output of the program (bottom right in Figure 1).

	N	Age	Progr. experience	Bugs found
Experts	8	25.88 (3.94)	108.00 (22.22)	2.75 (1.04)
Novices	6	26.17 (6.08)	42.00 (14.70)	1.50 (0.55)
t (12)		.109 (p=.915)	6.29 (p=.00004)	2.67 (p=.020)

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).

While some previous analyses of visual attention patterns approached similar situation with a long-term perspective, the main difference between the present



```
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, [illegible]le");
        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**
```

Fig. 1. A screenshot of IDE employed in the study, with 1 second of gaze overlaid.

study and previous analyses is the level of detail. To deal with the complexity of the programming process, the whole ten minute debugging sessions were divided into five two-minute segments. Our motivation for doing so was to explore the temporal properties of the eye-tracking data with a hope to better capture the underlying processing. Proportional fixation time (PFT) for each area was computed as a ratio of fixation time on an area to the overall fixation time on all areas. Number of switches per minute was computed as sum of all changes in visual attention focus between any of the three main areas during each segment per minute.

Results and discussion Table 2 and Figure 2 present the distribution of proportional fixation times. For the subsequent analyses of this measure, only data from code and output were used, because the proportional fixation times for code and visualization were almost perfectly negatively correlated (r (5) = -.971, p = .006). Thus, a 5 x 2 x 2 (segment, area, experience) ANOVA was conducted and revealed the main effect 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 found 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 Bonferroni adjustment procedure for multiple comparisons. This showed that while proportional fixation times during the last two phases were almost equal, the PFT during first two minutes was significantly different than during second segment (p = .037) and nearly significantly different than during fourth segment (p = .053). Although noticeable, the difference between second and third segment was not significant.

	Novices			Experts		
Segment (min.)	Code	Visualization	Output	Code	Visualization	Output
0-1	64.17	32.49	3.34	74.73	22.88	2.39
2-3	91.13	6.61	2.25	97.29	1.64	1.07
4-5	63.27	27.61	9.12	85.11	10.75	4.14
6-7	87.14	9.87	2.99	88.41	2.67	8.93
8-9	81.53	16.91	1.56	83.80	6.28	9.91

Table 2. Proportional fixation times (%) of Novices and Experts during the five segments of debugging.

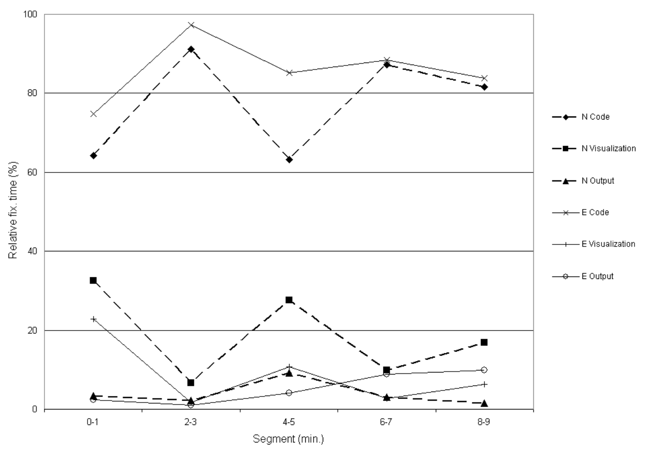


Fig. 2. Plot of proportional fixation times during five phases of debugging.

While previous quantitative investigations of eye-movement patterns of less experienced and expert programmers showed no differences between the visual behavior of the two groups [11], this finding seems to contradict the past results. Our analysis revealed an effect of experience on proportional fixation times.

Overall, throughout the whole debugging session expert programmers – who also found more bugs – relied more on the textual representation of the program than the less experienced programmers did. Output of the program became more important than visualization at later phases of the debugging strategies of experts, while novice programmers tended to rely on the visualization.

Table 3 and Figure 3 present the switching behavior in terms of *overall* number of switches per minute during the five segments of debugging. A 5 x 2 (segment, experience) ANOVA revealed 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).

Similarly as with the analysis of PFT, the significant main effect of segment on the switching frequency was analyzed using Bonferroni adjustment for multiple comparisons. 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.

	Novices		Experts	
Segment (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 3. Switches per minute between any of the three main representations during the five segments of debugging.

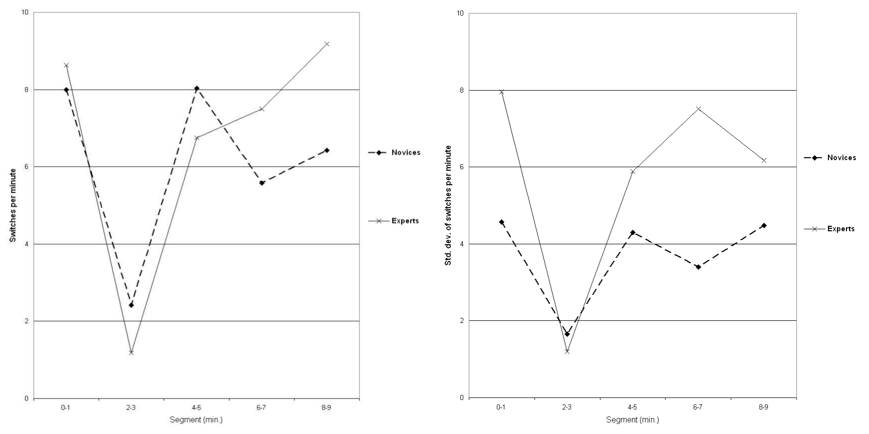


Fig. 3. Plots of switching behavior ((a) mean, (b) standard deviation) during five phases of debugging.

Finally, similarly as in [13] we have examined the type of switch programmers exhibit during debugging. Three types of switch were possible to perform during debugging: a switch between code and visualization (or back), a switch between code and output (or back), and a switch between visualization and output (or back).[1]Table 4 provides an overview of a breakdown of the switching frequency from Table 3 into the tree types of switches.

A 5 x 3 x 2 (segment, switch type, experience) ANOVA revealed significant main effect 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 switching frequency. Effect of experience was not significant (F(1,12) = 0.18, ns).

Interactions of segment and experience and type of switch and experience were not significant. However, interactions of segment and type of switch (F(8,96) = 4.75, p < .001, η^2 = .284) and segment, type of switch and experience (F(8,96) = 4.82, p < .001, η^2 = .286) were significant.

Pairwise comparisons using Bonferroni corrections showed that the main effect of switch type was due to the switch between code and visualization being significantly most 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).

	Novices			Experts		
Segment	Code - Visualization	Code - Output	Visualization - Output	Code - Visualization	Code - Output	Visualization - 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 4. Switches per minute for each of the three main types of switches during the five segments of debugging.

Segmentation of the eye-tracking data allowed us to analyze how the fixation patterns of programmers developed during the debugging. Visual analysis of Figure 2 and Figure 3 show saw-like patterns of visual attention in time, especially for the novice group throughout the whole process. Therefore, it can be assumed that the use of each representation of the program is not constant during the process, but oscillates between 64% of time to up to 97% of time (see Table 2. Number of switches and proportional fixation time on code were negatively correlated (r(5) = -.814, p = .093). Therefore, at times when the textual representation is being used the most, programmers tended to not to switch often between different representations.

[1] There are, in fact, six types of switches. However, to simplify the current analysis we have considered any switch between two representations as belonging to one type.

Both novice and expert programmers made most use of the visualization at the beginning of the process. In the next phase of debugging, both groups concentrated on the textual representation of the program, while decreasing the visual coordination activity. In the middle of the debugging process, novice programmers again paid much more attention to visualization and to output, and switched more frequently than in previous phase. Experts began to attend also to the output of the program and switch their visual attention between the three representations.

From the fourth phase, the expert group changed behavior so that we have observed frequent switches between output and program code. The plots of novices' PFT and switching behavior, on the other hand, continued in the saw-like pattern until the end of the debugging. At the final stage of debugging, expert programmers coordinated the three representations with the highest frequency of switches.

In summary, the high variance presented in the eye-tracking data of expert programmers indicates their diverging strategies, especially toward the end of the session. What the visual attention tracking data seem show, however, is that novice programmers engage in and alternate between two distinct approaches to coordinate the code and graphical representation. They begin by high frequency of switching between code and visualization. After this phase, they attend mostly the code, while exhibiting low number of attention switching to other representations. Once the code reading is finished, again, novice programmers change their strategy to the high-frequency attention switching.

4 General Discussion

Programming alone is a complex task to study. When these tasks take place within a rich and dynamic context, the analysis and interpretation of the resulting eye-tracking data present a serious challenge. Experimentation in software engineering is difficult and carrying out empirical work is complex and often time consuming [16]. This seems to be especially true in conducting and analysing of experiments employing eye-tracking to study software comprehension.

Program debugging and comprehension involve a variety of strategies that a programmer has to exercise to create a coherent mental model of a program. Modern programming environments present program-related information in multiple windows, and use graphics and animation to visualize some aspect of the program. This presents a challenge to the users who have to coordinate the representations by active selection of what information they want attend to. To get deeper insights into the processes as they are carried out in the presence of multiple linked representations, we have employed an eye-movement tracking technique to study visual attention patterns of expert and novice programmers.

To avoid manual extraction of eye-tracking data we made use of automatic techniques to reduce the data into eye-tracking measures. The relation of the eye-tracking measures to the comprehension processes, however, is not a straightforward process. In our previous studies in programming that employed visual attention tracking, we began to approach the task by studying effects of expertise on the eye-tracking patterns.

In contrast with some previous similar investigations, however, we further segmented the whole process into shorter sections to obtain finer level of detail about the process. The resulting eye-tracking metrics were analyzed using quantitative statistical methods, and plotted against time.

Using the proposed approach to eye-tracking data analysis, our results show how eye-movement patterns develop during debugging with multiple representations. Not surprisingly, programmers mostly visually attended source code, a confirmation of findings of some previous studies [13]. Our results extend on these findings and show how the representation use *developed* in time during debugging.

While we have not found any prevailing trends in the visual attention patterns that would reflect increasing or decreasing changes in use of the main representation, by segmenting the process into shorter intervals, we discovered temporal sensitivity of the visual attention patterns. In particular, we have discovered a saw-like pattern of use. We found that although there was a variance in the strategies, more experienced programmers change their strategies during debugging and focus their attention to output of a program at later stages of the process. While the results related to switching frequency show that for most of the debugging process the switching was not sensitive to expertise, toward the end of the ten minutes session more experienced programmers gradually exhibit higher frequency of switching. Based on these findings, we tend to believe that the changes in eye-tracking measures reflect both the importance of different representations during programming processes and differences in debugging strategies.

Some of the most intriguing aspects of visual attention behavior, however, cannot be discovered only using pure quantitative reductionist approaches to eye-tracking data analysis. We have illustrated that the temporal aspects of eye-tracking data during programming can provide valuable insights about the representation use. There are, however, also issues that limit the potentials of automatic methods to analyze the gaze patterns and relate them to the underlying processing. In particular, arranging participants into groups – whether based on experience or time – smooths away the individual differences. In [8] the two most similar scanning patterns while reading an algorithm belonged to subjects from opposite experience groups. Also in our study the individual differences sometimes seemed to predominate over a stereotypical group behavior. This caused the variability within a group, for example, in eye-tracking data related to switching behavior; while the measure-means of the two groups were similar, the behavior of more experienced group contained greater amount of variance. These variances, in turn, seem to impair the traditional quantitative approaches to variance analysis in their assumptions of homogeneity of variances.

Therefore, in parallel to automatic quantitative methods, we are investigating the potentials of qualitative approaches to eye-tracking data analysis in dynamic programming environments. These approaches to visual attention data during programming are indeed required to complement the quantitative view on the process. While quantitative methods can help us to discover interesting events in visual behavior, we suggest that more qualitative approaches shall be employed to provide detailed explanations.

Our study also raises several issues that need to be addressed by future research into the link of eye-tracking data and processes involved in programming. In the present study, the segments of the data were delimited based on fixed time-interval. Although the time-based approach allows for clearly defined segmentation, different participants seem to proceed with debugging at their own pace. For example, some expert programmers found bugs faster, and therefore might have changed their strategies sooner than programmers who have not found bugs. Therefore, aggregating and regarding individual behaviors at certain fixed interval as representing a group behavior might be problematic.

To study individual behavior, however, boundaries based on better defined subtasks and events shall be determined as references to behavioral units. For example, one class of such delimitations can be related to a programmer finding a bug, or changing a strategy. It can be then possible to examine, for instance, how users modify their strategy after a bug has been found. We plan to investigate this idea in future.

Another interesting observation that fuels our future research of aspects of visual attention in programming is related to switching behavior. While most of the visual attention switches performed during debugging were balanced and between two representations, we have observed that more experienced programmers during certain phases of debugging tend to exhibit switches that coordinate three representations. The transition matrix representing the switching behavior then becomes asymmetrical (e.g. there are more switches from code to visualization than from visualization to code). We propose that a degree of asymmetry of the transition matrix could be a new higher-level eye-tracking metric.

5 Limitations and Conclusions

The main limitation of this study can be seen in the low number of participants. However, the main focus of this study was on the methods to analyze the eye-tracking data during programming rather than on testing the hypotheses related to the use of visualization in programming. As there are no automatic tools to perform the proposed analysis, we began with a reasonable yet illustrative sample size. Any further quantitative investigations shall consider recruiting more participants to exhaustively test the hypotheses set by the present study.

In summary, our exploratory study shows that segmentation of eye-tracking data in general seems promising. We have attempted to segment the data set according to time into shorter segments of equal duration, one of many potential approaches to segmentation. Consequently, both the proportional fixation time and switching frequency showed sensitivity to the effect of different phases of a debugging session.

Contrary to previous findings that approached the eye-tracking measures from a long-term reductionist perspective, our analysis also revealed differences in representation use during debugging. Our findings indicate that experts exerted more efforts to integrate the information available and changed their visual strategies during the process, in particular to relate code and output information at the later stages of debugging. Novice programmers, on the other hand, seemed to alternate between two strategies without being able to modify their approach.

Acknowledgment

The feedback from three anonymous reviewers was valuable and has materialized as improvements of this paper.

References

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

[2] Blackwell, A.F., Whitley, K.N., Good, J., Petre, M.: Cognitive factors in programming with diagrams. Artif. Intell. Rev. **15** (2001) 95–114

[3] Goldberg, J., Kotval, X.P.: Computer Interface Evaluation Using Eye Movements: Methods and Constructs. International Journal of Industrial Ergonomics **24** (1999) 631–645

[4] Cowen, L., Ball, L.J., Delin, J.: An eye-movement analysis of web-page usability. In Faulkner, X., Finlay, J., Détienne, F., eds.: People and Computers XVI: Memorable yet Invisible: Proceedings of HCI 2002, Springer-Verlag Ltd (2002)

[5] Rayner, K.: Eye movements in reading and information processing: 20 years of research. Psychological Bulletin **124** (1998) 372–422

[6] Reichle, E.D., Pollatsek, A., Rayner, K.: E-Z Reader: A cognitive-control, serial-attention model of eye-movement behavior during reading. Cognitive Systems Research **7** (2006) 4–22

[7] Goldberg, J., Wichansky, A.: Eye Tracking in Usability Evaluation: A Practitioner's Guide. In Hyönä, J., Radach, R., Deubel, H., eds.: The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research, Elsevier Science (2003) pp. 493–516

[8] Crosby, M.E., Stelovsky, J.: How Do We Read Algorithms? A Case Study. IEEE Computer **23** (1990) 24–35

[9] Romero, P., du Boulay, B., Lutz, R., Cox, R.: The effects of graphical and textual visualisations in multi-representational debugging environments. In: HCC '03: Proceedings of the IEEE 2003 Symposia on Human Centric Computing Languages and Environments, Washington, DC, USA, IEEE Computer Society (2003)

[10] Nevalainen, S., Sajaniemi, J.: Short-Term Effects of Graphical versus Textual Visualisation of Variables on Program Perception. In: Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05), Brighton, UK (2005) 77–91

[11] Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: Analyzing Individual Differences in Program Comprehension with Rich Data. Technology, Instruction, Cognition and Learning **3** (2006) 205–232

[12] Bednarik, R., Tukiainen, M.: An eye-tracking methodology for characterizing program comprehension processes. In: ETRA '06: Proceedings of the 2006 symposium on Eye tracking research & applications, New York, NY, USA, ACM Press (2006) 125–132

[13] Romero, P., Lutz, R., Cox, R., du Boulay, B.: Co-ordination of multiple external representations during java program debugging. In: HCC '02: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), Washington, DC, USA, IEEE Computer Society (2002) 207

[14] Bednarik, R., Tukiainen, M.: Visual attention tracking during program debugging. In: NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction, New York, NY, USA, ACM Press (2004) 331–334

[15] Bednarik, R., Tukiainen, M.: Validating the restricted focus viewer: A study using eye-movement tracking. Behavior Research Methods (in press)

[16] Basili, V.R., Shull, F., Lanubile, F.: Building knowledge through families of experiments. IEEE Transactions on Software Engineering **25** (1999) 456–473