UNIVERSITY OF JOENSUU
COMPUTER SCIENCE AND STATISTICS
DISSERTATIONS 21

Roman Bednarik

Methods to Analyze Visual Attention Strategies: Applications in the Studies of Programming

Academic dissertation

To be presented, with the permission of the Faculty of Science of the University of Joensuu, for public criticism in the Louhela Auditorium of the Science Park, Länsikatu 15, Joensuu, on December 15th 2007, at 12 o'clock.

UNIVERSITY OF JOENSUU
2007

Supervisors	Professor Markku Tukiainen Department of Computer Science and Statistics University of Joensuu Joensuu, FINLAND
	Professor Erkki Sutinen Department of Computer Science and Statistics University of Joensuu Joensuu, FINLAND
Reviewers	Professor Kari-Jouko Räihä Department of Computer Sciences University of Tampere Tampere, FINLAND
	Associate Professor Dario Salvucci Department of Computer Science Drexel University Philadelphia, USA
Opponent	Associate Professor Andrew T. Duchowski School of Computing Clemson University Clemson, USA

ISBN 978-952-219-067-3 (printed)
ISBN 978-952-219-068-0 (PDF)
ISSN 1796-8100 (printed)
ISSN 1796-8119 (PDF)
Computing Reviews (1998) Classification: H.5.2, H.5.1, I.3.6, H.1.2, D.2.2

Joensuun yliopistopaino

Joensuu 2007

Methods to Analyze Visual Attention Strategies: Applications in the Studies of Programming
Roman Bednarik

Department of Computer Science and Statistics
University of Joensuu
P.O.Box 111, FIN-80101 Joensuu, FINLAND
`roman.bednarik@cs.joensuu.fi`

University of Joensuu, Computer Science and Statistics, Dissertations 21
Joensuu, 2007, 188 pages

Abstract

The main problems identified by the research that employs visual attention tracking to retrospectively evaluate user strategies are low efficiency and difficulties of the methodology in interpreting the data. In this thesis we attack these problems in the domain of multirepresentational dynamic programming environments. Firstly, by replicating a previous experiment in debugging, eye-tracking was shown to be a superior technique for tracking the visual attention of programmers when compared to the Focus Window Technique. Traditional eye-tracking measures and the approaches used to analyze them are, however, shown to be only partially able to describe all the subtle behaviors of visual attention during programming. Thus, secondly, alternative methods and new ways of processing eye-tracking data and measures are presented. We suggest the eye-tracking studies should utilize a) the segmentation of eye-tracking data sets, both temporally and spatially, b) a binomial test when dealing with the sparse eye-tracking data sets, c) a visualization of the data on multiple levels of detail, and d) employ high-level contextualized measures. Thirdly, by applying these approaches, a more detailed and novel view of the strategies of novice and expert programmers is presented. Expert and novice programmers' visual strategies are described and shown to differ in several aspects. Expert programmers are better able to integrate more information from the representations available. In particular, they concentrate more on code at the beginning and on relating code to output at the later phases of the process. Novice programmers engage in a limited range of strategies that they seem to regularly alternate between; in particular, they either switch their visual attention frequently between code and graphical representation or they tend to mainly focus on one of them. Finally, implications of the findings to the field of eye-tracking methodology are drawn. These consist of, for example, a call for better automatic analysis tools that facilitate the techniques presented here.

Keywords: eye-tracking, psychology of programming, visual attention, methodology

Acknowledgements

I did not know that writing the acknowledgment section can easily became so difficult. In a couple of lines I wished (and still do) to thank all of those who helped me; but along the ragged road there were simply too many of them to be mentioned here.

I thank Markku Tukiainen for giving me the opportunity to work with him, sharing a passion to better understand the people, computers and their interactions, and for his infinite trust, support and guidance. Erkki Sutinen deserves a word of thanks for the opportunities he gave me and for teaching me the lessons of bravery, devotion, thoughtful nuttiness and kindness.

I am indebted to my colleagues: to Niko Myller for being a diligent coauthor and for the constant and instant feedback on my rambles, ideas and doubts; to Justus Randolph for never slowly but always surely giving me precisely aimed advices, and for being a coauthor and language editor of my work. Many of my colleagues deserve my thanks, spending the working days with you is a pleasure and fun.

I thank Kari-Jouko Räihä and Dario Salvucci for agreeing being the reviewers of my work, and providing helpful critique and comments. I am honored and thankful to have Andrew Duchowski as an opponent.

This research has been supported in parts by the Faculty of Science, University of Joensuu, Finland, by the East Finland Graduate School in Computer Science and Engineering (ECSE), Finland, by the Emil Aaltosen Foundation, Finland, by the Tekniikan edistämissäätiö (TES), Finland, and by the Psychology of Programming Interest Group (PPIG). A part of the research and writing has been conducted at the University of Pittsburgh, USA, and at the University of Kabarak, Kenya.

I thank to my dear ones, my family and my wife Kaisa, for what cannot be expressed here in writing.

Joensuu, November 2007
– Roman Bednarik

List of original publications

This thesis consists of an introduction and the following peer-reviewed papers, which were reproduced here with permission:

P1. Bednarik, R., Tukiainen, M.: Validating the Restricted Focus Viewer: A Study Using Eye-Movement Tracking. *Behavior Research Methods*, 39(2), 2007, pp. 274-282.

P2. Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: Effects of Experience on Gaze Behaviour during Program Animation. In Proceedings of the *17th Annual Psychology of Programming Interest Group Workshop (PPIG'05)*, Brighton, UK, June 28 - July 1, 2005, pp. 49-61.

P3. Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: Analyzing Individual Differences in Program Comprehension. *Technology, Instruction, Cognition and Learning (TICL)*, 3 (3-4), 2006, pp. 205-232.

P4. Bednarik, R., Randolph, J.: Studying Cognitive Processes in Program Comprehension: Levels of Analysis of Sparse Eye-Tracking Data. In *Hammoud, R. (ed): Passive Eye Monitoring: for Safety, Security, Communications, Medical and Web Applications. 2008, Springer.*

P5. Bednarik, R., Tukiainen, M.: An Eye-tracking Methodology for Characterizing Program Comprehension. In Proceedings of the *2006 Symposium on Eye Tracking Research and Applications (ETRA 2006)*, March 27-29, San Diego, CA, USA, ACM Press, pp. 125-132.

P6. Bednarik, R., Tukiainen, M.: Analysing and Interpreting Quantitative Eye-Tracking Data in Studies of Programming: Phases of Debugging with Multiple

Representations. In Proceedings of the *19th Annual Workshop of the Psychology of Programming Interest Group (PPIG'07)*, Joensuu, Finland, July 2-6, 2007, pp. 158-172.

Contents

Chapter 1

Introduction

THIS thesis aims to explore and promote understanding of the link between visual attention data and processes involved in programming by dealing with the methodological challenges of eye-tracking as applied in that domain.

Although the technological problems of eye-tracking systems are being continually resolved, granting the increasing usability of the technique, the methodological issues prevent it from being used further in a wider context. Apart from the somewhat remaining technical problems, Jacob and Karn (2003) list two methodological problems with eye-tracking: labor-intensive data extraction and difficulties in their interpretation.

Modern eye-tracking systems are easy to operate, do not interfere with participants, and are able to capture a large proportion of the population. Commercially available eye-tracking systems are often supplied with an automatic recording and analysis software, attempting to reduce the labor of data extraction. While this automation can facilitate the analysis for static computer interfaces and relatively simple tasks, the studies of complex user processing and interactive systems present a new challenge to eye-tracking research.

In many cases the dynamics of the scenes being presented to participants – such as in modern computer programming interfaces – makes it hard for an eye-tracking researcher to easily 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 inefficient, may bring about several unwanted outcomes. For example, the manual annotation of the fixations might pose a threat to the accuracy of the data analysis.

Another problem with the application of eye-tracking in evaluating interfaces is that of interpreting and relating the extracted data to the underlying processing. In a typical eye-tracking study, the analysis tends to be quantitative; it starts from selecting the eye-tracking measures, continues through delimiting the scene into the areas of interest, aggregating the measures with respect to the areas, and ends with linking the observations to the phenomena in question. In the studies that investigate cognition with interfaces that present text, graphics, and dynamic content, the analysis phase of an eye-tracking experiment might become the most daunting task of the whole investigation. In addition, to preserve the experimental validity of an eye-tracking experiment, the researcher might not wish to impose constrains on the participants' behavior or to employ artificial tasks and environments. Balancing user freedom, the validity of a study, and the constraints imposed by the study settings can further increase the complexity of the analysis; the validity factors include, for example, the presentation of information in several dynamic windows, the freedom of users to select when and what information they want to see, or the possibility of the system to present interaction dialogues such as questions. As the complexity of the interaction and cognition increase, the link between the eye-tracking data and underlying processing becomes more difficult to study.

This thesis describes research where visual attention tracking has been employed to study human processes involved in program comprehension as performed with the aid of multiple representations of a program. To comprehend a program, a programmer needs to construct a mental model of what the program does and how it does it; program comprehension is thus an integral part of any software development process. Program comprehension is also a cognitively complex skill that is difficult to acquire. Thus, the question of how a novice becomes a skilled professional is one of the central inquiries in studies of the psychology of programming. Although originally centered on professional programmers developing computer programs, studies of programming now extend beyond these borders (Blackwell, 2002), both in terms of users and application domains.

There are several possibilities to get insights into the behavior and strategies exhibited by programmers interacting with a computer. In similar research situations where the stimuli are visual and the user reasoning is related – or even dependent – on it, eye-tracking systems have been shown to be useful in revealing the patterns of visual attention during the task. The classical examples of applications of eye-tracking include pioneering studies relating the eye-movement patterns to cognitive processes (Just and Carpenter, 1976), research in reading (Rayner, 1998), or studies investigating differences between novices and experts in terms of the eye-movement patterns (e.g., Hyönä et al., 2002, Law et al., 2004).

In studies of programming, instead, investigations have been primarily based on verbal protocols, a well established – and probably the most popular – method used to capture and analyze human information processing. Despite their potential, visual attention tracking methods have not been applied widely in this domain, but they are slowly finding their place in the toolboxes of the researchers who study the psychology of programming (Crosby and Stelovsky, 1990, Stein and Brennan, 2004, Romero et al., 2002b, Nevalainen and Sajaniemi, 2004, 2005). At the moment, however, little is understood regarding the visual strategies of programmers and how to study them using eye-tracking.

1.1 Background and Motivation

Program development normally takes place within integrated development environments. These tools often present the information related to the program being developed using multiple representations of the program. Several important questions related to visual attention and its role during programming within these environments can be raised. A general question about what information sources programmers attend to when working with a development environment leads one to first ask about how to record the visual attention in programming. Whether and how the cognitive processes involved in programming are reflected in visual attention patterns, however, is not completely understood. Are there general patterns of visual attention with which programmers attend the source code and the adjacent representations while comprehending the program? What are the programmers' visual strategies and how can they be identified from eye-movement data? Does the focus of visual attention correlate with other information about the comprehension process? Is it possible to distinguish between good and poor comprehension based on the information about visual attention?

The lack of knowledge about these and related aspects of visual attention during programming motivates the research presented in this thesis. Eye-tracking technology seems to be a suitable tool to increase our understanding about the role of visual attention in programming and, therefore, the possibilities and limitations of it and the associated techniques need to be studied and understood.

The main motivation of this research is therefore a methodological one. As it will become clear later, several methodological issues have been unearthed that need to be considered before visual attention tracking – and eye-tracking in particular – can be employed to study programming strategies in a meaningful way. For instance, eye-tracking data come in large volumes, a property that makes them difficult to process and directly link to the underlying processes. Previous eye-tracking research has therefore developed techniques and measures that facilitate the analysis of eye-tracking data in studies of usability. For example, Goldberg and Kotval (1999) proposed a set of eye-tracking measures that allow for the automation of the process. Thus, the large quantities of raw eye-tracking data can be significantly reduced to make the analysis of the data considerably more efficient. However, determining what measures to use and how to interpret the data and relate them to the underlying processing or to usability aspects are tasks not well understood yet.

Linking eye-tracking data to underlying cognitive processes has become the primary challenge in eye-tracking studies. For instance, in the eye-tracking studies of usability, Jacob and Karn (2003) argue that the challenge has been *"probably the single most significant barrier to the greater inclusion of eye tracking"*. There are several reasons contributing to this challenge, such as the aforementioned dynamic nature of modern computer interfaces, the volumes of eye-tracking data, the complexity of the tasks being studied, the sparseness of the data sets, other human factors and the concerns of researchers to maintain high levels of validity, to name just a few.

At the moment, little is known regarding how to apply eye-tracking to study programming activities and about how to evaluate user strategies when dynamic graphical interfaces are employed in programming. Therefore, we[1] believe it is high time to consider the methodological issues of eye-tracking as applied in this domain.

1.2 Focus of Research and Research Questions

This thesis is primarily concerned the with methodological aspects of visual attention tracking studies of programming activities. In particular, it intends to contribute to the understanding of how to record programmers' visual attention patterns, and how to analyze and interpret the resulting data. At the same time, this thesis expands the knowledge obtained in the previous studies of visual attention during reading algorithms (e.g., Crosby and Stelovsky, 1990) to real-world situations where programmers have to deal with multiple representations and dynamic displays. To summarize the focus of the thesis: we investigate how to obtain and make sense of visual attention data in the context of programming with multiple representations and then we apply these methods to improve the understanding of the processes involved in programming.

This work does not specifically aim to answer issues related to cognition during programming *per se*; it seeks to inform the studies of programming or problem-solving about how to incorporate visual attention tracking. Nevertheless, by applying the knowledge of how to conduct, analyze, and interpret eye-tracking data in programming studies, the work reported in this thesis presents new insights into how programmers divide their visual attention. Finally, because the research situations in this work closely correspond to those of eye-tracking studies of usability, these and similar studies will also benefit from the findings.

[1]Although this thesis is written by a single author who coordinated the research, the plural subject pronoun is used throughout this thesis because the research has not been conducted in isolation and often there were several contributors.

We approached the development of an eye-tracking methodology for studies of programming by answering the following questions, which are sorted into three main research areas: 1) tools, 2) measures and strategies, and 3) analysis approaches.

1. An alternative tool to track visual attention, the Restricted Focus Viewer, has previously been applied to study programming behavior. How does it compare to a table-mounted remote eye-tracking device?

2. What are the characteristics of programmers' visual attention strategies while working with multiple representations and how does one identify them from the eye-tracking measures? For example, are expertise levels of programming reflected in the eye-tracking measures?

3. What are the limitations and advantages of the current approaches to the analysis and interpretation of visual attention data during programming and how they can be improved so that they provide better information about the underlying processes?

1.3 Method

The work reported after this introduction attempts to answer the research questions mentioned above. Focusing on the advancement of the methodological aspects of eye-tracking, it borrows from multiple research paradigms and methods. At the heart of the thesis lie two empirical studies. These studies – conducted under similar research settings as eye-tracking studies of usability – serve two purposes. On their own, the studies were designed to answer specific research questions and they produced new knowledge about the aspects of visual attention strategies in programming.

In addition, the two experiments can be seen as exploratory case studies because they provided the initial sources of eye-tracking data for subsequent analyses. These analyses were performed in cycles to further investigate the potentials of the methods to characterize the links between the underlying processes and overt visual attention during programming activities. The outcomes of these analyses are critically discussed and serve as rationale for proposing improvements or alternative ways to analyze and interpret the data. Using the new approaches, the work returns back to the original studies and applies the new perspectives to discover new knowledge about the strategies involved in programming.

Therefore, a part of the work in this thesis resembles the studies conducted within the constructive research paradigm (Lukka, 2003), in how it iterates between the implementation of a method and its evaluation. The implementation and construction phases can be seen as applying both the current methods and as proposing alternative approaches and their improvements. The improved methods are the constructs and the evaluation takes the form of analytical reflections on the findings achieved using the established and the proposed approaches. As new findings come to light, these are analyzed and the proposed solutions are integrated into the next phase of the research. Therefore, in the way the different approaches are employed and combined to gain understanding and knowledge, and in the analysis of what works in practice, this research can also be seen as drawing from the pragmatic position to science (Creswell, 2003).

Table 1.1 presents a summary of methods employed to answer the research questions.

R.Q.	Method	Paper(s)	Section(s)
1	Experimental, Replication	P1	4.1
2	Experimental	P2	4.2
2,3	Experimental	P3	4.3
3	Critical literature review	P4	4.4
2,3	Case study	P5	4.5
2,3	Case study	P6	4.6

Table 1.1: Summary of methods employed to answer the research questions (R.Q.), their respective publications, and the sections of this thesis that discuss the topic.

1.4 Research Process

The initial purpose of the thesis was to investigate the role of visual attention during programming tasks with multiple representations and apply the available methodologies to track and analyze visual attention. In the beginning, we had applied the methods and tools – as it had been done in the previous and related research – to help improve the understanding of the role of visual attention in programming.

During the research process, however, new challenges in applying the traditional approaches appeared that forced us to expand the focus of the study to also investigate the weaknesses, limitations, and possibilities of visual attention tracking techniques and methods. The studies reported in papers (**P1**) and (**P2**) serve as the source of empirical data for the reports in (**P6**) and (**P3**, **P5**), respectively. Methodological improvements are reported in (**P1**, **P4**, **P6**) and in Chapter 4.

Chapter 4 re-examines the findings from a new perspective and proposes solutions to the problems encountered. The improved methods and measures allow for more accurate description of visual attention strategies. The knowledge about the strategies gradually increases and becomes more focused, as reported in all publications except for (**P4**).

1.5 Research Approaches and Analysis Methods in Human-Computer Interaction and Eye-Tracking Research

Research methods in the behavioral and social sciences, according to the definition of McGrath (1995), are the tools used to gather and analyze information. Human-Computer Interaction (HCI), due to its multidisciplinary background, draws on multiple research theories, approaches, and methods (Carroll, 2003). Of the set of empirical methods, experimental approaches, cognitive modeling and task analysis, ethnography, fieldwork, or case studies are examples of the diverse approaches employed in modern HCI research.

Distinguishing how data about a phenomenon is gathered and analyzed, ethnographic methods or case studies typically provide and allow for qualitative interpretations of the phenomenon, while the other methods lend themselves to quantitative views. However, as Sharp et al. (2007) points out, assuming that a certain form of data gathering would result in either only quantitative or only qualitative data would be a fallacy.

Modeling is a line of HCI research that focuses on understanding user behavior and providing a formal description (i.e. the model) about some part of user interaction. Models can be used, for example, to predict skilled user performance, behavior, and cognition during a task. Other families of models focus on predicting human movements during a specific interaction task. Two recent accounts provide an overview of modeling in HCI: MacKenzie (2003) presents an overview of predictive and descriptive models; and John (2003) presents an overview on the models based on human information processing.

Based on psychological research, the experimental approaches in HCI mainly provide quantitative data about a phenomenon. The often cited advantages of involving experimentation are the objective evaluation of human performance, measurement precision, and rigorous control of the research situation. Quantitative data can be evaluated using statistical procedures to make inferences about the relations between the sampled population and researcher-controlled variables.

Experimental approaches, however, have also been criticized, among other problems, 1) for limiting HCI research by ignoring the variations in and effects of individual behavior, 2) for being susceptible to confounding variables that are hard to control, 3) for de-contextualization, and 4) for focusing too much on the casual relations between the variables. The emphasis of experimental approaches on objective measurement and on strict control limits the range of behavior that can be studied (Miller, 1984). In particular, controlled experiments reduce the situation to a number of measurable independent and dependent variables. The reductionist approach can, for instance, be seen in the studies of psychology of programming.

In those studies, for example, a performance measure – such as an average time of a group of participants spent on a programming activity – can be related to some of the characteristics of that group. Monk et al. (1993) call this lack of focus on individual behavior as not paying "*due attention to the richness (and fine detail) of human behaviour*".

The limitations of experimentation in HCI motivated the shift to ethnographic approaches (Ormerod et al., 2004). Ethnographic approaches, in contrast to controlled experiments, typically provide qualitative data collected in a naturalistic settings. Trying to cause as little distraction to the observed participants, an ethnographic researcher investigates the participants who are already engaged in carrying out their natural tasks. The goal of the ethnographic study is to gain a holistic understanding of phenomena. At the moment, there seems to be an ongoing discussion about the role of ethnography in HCI research (Räsänen and Nyce, 2006).

What are the ways to best make use of such diametrically opposing methods? Monk et al. (1993) discuss the tensions between experimentally rooted research and ethnographic approaches to the study of computer-mediated communication. They claim, for instance, that taking either of the two approaches to interpret the same data can lead to different interpretations of the data. While arguing for more ethnography-based research in HCI, in their attempt to reconcile the proponents of these two approaches Monk et al. (1993, p.4) say that "*Both ethnographic and experimental methods have their place in the study of human thought and behavior. The problem is to select the right method, and then to apply it correctly.*"

While selecting the right method might be difficult, mixing different methods seems to be a common practice in HCI research (Ormerod et al., 2004). Examples of this approach can be seen in the work of Ormerod et al. (2004), integrating ethnography and experimentation, or in the study of Murphy et al. (1999) combining a case study approach and an experiment. Through the integration of multiple perspectives, a deeper understanding of a phenomenon on one hand, and objective evaluation of it, on the other hand can, with hope, be achieved.

It shall then come as no surprise that the current situation in research practice is not anymore a debate of qualitative versus quantitative (Creswell, 2003). Instead, mixed approaches are employed "*... because they work to provide the best understanding of a research problem*" Creswell (2003, p.12).

Using the terminology of McGrath (1995), methods (called modes of treatment) are the "*ways by which a researcher can deal with a particular feature of the human systems that are to be studied.*" Measurement methods, a subset of modes of treatment, are techniques to determine what value, state, or level the feature of interest has. Eye-movement tracking (hereafter eye-tracking) – and similar tools such as display-masking techniques – then can be seen as measurement techniques for determining a feature of human visual system called visual attention.

Which methods of data collection and analysis to use are the central questions in research design (Creswell, 2003). In past, eye-tracking has been used as a data collection technique for various methods of HCI, including modeling and evaluation.[2] As it will be shown later, a great number of past retrospective eye-tracking studies that evaluated user strategies take the reductionist approach to data and measurements. This approach, which is supposed to capture, describe, and inform about the user visual strategies, however, might be insufficient in more complex problem-solving situations, such as debugging a program.

At the present, eye-tracking is considered to be a standard tool of an HCI analyst (Renshaw et al., 2006). Yet, how to make use of the data to maximize the benefits of the technology for evaluation purposes is not well understood. In this thesis we therefore investigate one of the central questions in eye-tracking research. We aim to apply the current approaches methods to eye-tracking analysis, identify their methodological challenges, and suggest ways to address these problems.

1.6 Organization of the Thesis

This is a multiple-paper thesis that consists of an introduction and original research articles. From the next chapter, we give an overview of the current situation in two research areas that underpin the present work – that is the studies that used eye-tracking for retrospective evaluation of computer displays and user strategies, and the studies of psychology of programming. In the former, the state-of-the-art of data analysis and interpretation methods, as conducted in recent studies that employed eye-tracking data to evaluate user strategies with computer displays, is presented. In the latter, the methodologies employed to the study behavior of computer programmers, and the recent studies that used eye-tracking for doing so are overviewed. After the overviews, the contribution of this thesis is reported as a retrospective discussion of the research process and its results. This general discussion complements the results and discussions in each of the original publications. Next, summaries of the main findings and results are presented. The thesis concludes with an outline for future work.

[2]The other mainstream of eye-tracking research, identified by Jacob and Karn (2003), is using eye-tracking as a real-time input device. In this thesis we do not consider the methodological challenges stemming from this research direction.

Chapter 2

Visual Attention and Eye-Tracking Methodology

As the eyes of the reader of this thesis dart over the text of this chapter, a set of processes is orchestrated that produce the resulting mental representation of the meaning of the text. Because parafoveal reading is difficult, a new region of text needs to be fixated, encoded, and comprehended.

Studies of reading have greatly benefited from knowing where the eyes fixate at during reading (Rayner, 1998), and visual attention tracking methods have been employed to mediate that information. Reading, however, is a well-defined task and also a task that is much more constrained than, for example, visual search (Goldberg and Wichansky, 2003) or some other complex problem-solving tasks with modern dynamic displays.

Besides reading (e.g., Just and Carpenter, 1980, Rayner, 1998), eye-tracking has been employed in a wide range of studies and application areas, including gaze-based interaction (e.g., Jacob, 1991, Jacob and Karn, 2003), eye typing (Majaranta and Räihä, 2002), menu selection (Crosby and Peterson, 1991, Aaltonen et al., 1998), usability (Goldberg and Kotval, 1998, 1999), driving (Underwood et al., 2003), or in virtual reality (Duchowski et al., 2000, 2002). These and other investigations show that there are several good reasons for employing eye-tracking as a research methodology to investigate visual attention patterns. Among the many advantages for applying eye-tracking to study human behavior, participants in eye-tracking studies do not have to be trained to exhibit their natural strategies (as they need to be trained when thinking aloud). In addition, most of the current eye-trackers are highly non-invasive. These properties make eye-tracking a usable, non-intrusive tool causing no or little interference with natural behavior. At the moment, eye-tracking is considered to be a standard technique in HCI research (Renshaw et al., 2006).

In the following section, we first discuss the main assumptions upon which the applied eye-tracking research relies and then we focus on the current methodologies to track and analyze visual attention.

As do all vertebrates, humans have movable eyes. We move our eyes in order to bring an image of the inspected object onto the fovea, a small and high-resolution area of the retina where the cones are most densely packed. Once the image of the object is stabilized on the retina, the information can be extracted. This way visual attention is linked with the current gaze direction, and most of the time the visual information is processed the visual attention is also diverted to the point of inspection. In eye-tracking research, this principle is called *an eye-mind assumption* (Just and Carpenter, 1980). In a complex information processing task, the connection between the focus of attention and gaze direction is presumably tight (Rayner, 1998).

There are, however, situations when visual attention and gaze direction are dissociated; parafoveal or peripheral processing can be used to extract information from the environment. As Duchowski (2003, p.14) points out: "*...in all eye-tracking work...we assume that attention is linked to foveal gaze direction, but we acknowledge that it may not always be so.*" This thesis rests on the assumption of Just and Carpenter; however, similarly to Duchowski, we acknowledge that the focus of visual attention and the direction of gaze might be, at times, dissociated.

Because of the limited size of the fovea, the gaze has to be re-directed to the new locations of interest in the scene or object in order to allow for the new details to be perceived and processed. Once the eyes are directed and stabilized on the area of interest, the visual information can be extracted and encoded. The relatively stable position of the gaze direction is called *fixation*, while the shift of the gaze between two fixations is called *saccade*. A single saccade can last between 30 and 120 ms and can span over 1 to 40 degrees of visual angle (Sibert and Jacob, 2000), with velocities ranging up to 500 degrees per second (Rayner, 1998). It is assumed that a) during the saccades the vision is suppressed and no information is extracted and processed (this phenomenon is known as *saccadic suppression* (Matin, 1974)), and b) once initiated, a saccade's destination cannot be altered.

Fixations, on the other hand, are the movements of eyes during which information can be extracted, encoded, and processed. This principle of the immediate processing of information during a fixation is called *the immediacy assumption* (Just and Carpenter, 1980). Typically, the fixation duration ranges from 200 to 300 ms (Rayner, 1998) and is thought to be related to the processing required to extract and interpret the information (Just and Carpenter, 1976, Goldberg and Kotval, 1999).[1] Following the implications of the eye-mind assumption and the immediacy assumption, if we can track the movements of the eyes, we can also obtain insights into and investigate the path and focus of visual attention. Knowing which objects and elements have been visually fixated, in which order, with what frequency, and in which context, we can attempt to infer what cognitive processes were involved in performing a task related to these objects. Previous research has indeed firmly confirmed this relation among eye movements, visual attention, and underlying cognitive processes (e.g., Just and Carpenter, 1976, 1980, Rayner, 1998).

Other types of eye-movements exist (Carpenter, 1988, Duchowski, 2003). *Smooth pursuits* occur when the eyes track a moving object to match the velocity of the eyes with that of the target's and, therefore, reduce the target's retinal motion to minimum. *Nystagmus* eye movements serve to correct the motion of the head or to attend to repetitive patterns. *Miniature eye movements*, such as drifts and microsaccades, that might appear as a noise in the eye-movement signal are executed to stabilize the retinal image during a fixation. Finally, the movements of the eyes that are not conjugate, executed for instance when attending an approaching object, are called *vergences*. For a more detailed review of eye-movements and their models, we refer an interested reader to (Carpenter, 1988).

2.1 Visual Attention Tracking and Analysis

Accepting that the fixations and underlying processes are connected, how does one make use of that information? Jacob and Karn (2003) distinguish between two ways of using the gaze data from an eye-tracking device: real-time eye-tracking and retrospective analysis.[2] The former approach involves gaze location as a direct interaction medium, so that the eyes of the user have a direct impact on the interaction with the interface in real time.

The latter use of gaze-data – on which this thesis is primarily focused – as a retrospective analysis tool, is normally conducted in either a top-down way or in a bottom-to-top fashion. For studies based on a cognitive theory (using the top-down approach to data), Salvucci and Anderson (2001) propose relating the eye-movements to the predictions of a process model. However, there are many situations when the researcher approaches the analysis of eye-tracking data in a bottom-up way: the resulting data are investigated for common patterns and only then a model or a hypothesis about underlying cognitive processing is created.

This normally starts with a researcher defining so-called *areas of interest*, which

[1]There are numerous definitions of fixation in the literature. The working definition applied in this thesis is the one of Jacob and Karn (2003).

[2]Retrospective analysis is sometimes also referred to as diagnostic use of eye-tracking.

are usually rectangular static areas covering the interface elements in question. The researcher then conducts an experiment, records the gaze data, and, after the experiment, processes the data into measures, tries to relate the measures over the areas of interest to the manipulated variables and underlying cognitive processes. Therefore, it is the central task of the eye-tracking researcher to operationalize the behavior in terms of the eye-tracking measures, construct the measures from the recorded gaze points, and retrospectively relate the measures to the observed, hypothetical, or manipulated changes or interventions in the task and stimuli.

2.1.1 Eye-tracking Measures

Bower and Clapper (1989, p.298) pointed out that "*The main difficulty with eye-movement recordings [to study cognitive processes] is that the investigator is in danger of becoming buried in mounds of data as well as details of the technical apparatus. Therefore scientists considering the use of eye-movement recordings are advised to become familiar with the costs in time and money before they embrace such an expensive and data-rich source. Also they will need a set of practical data-reduction programs to help them deal with the huge volume of eye-fixation data generated by a few subjects reading just a few passages.*"

Goldberg and Kotval (1998, 1999) proposed a set of eye-tracking measures that allow for the reduction of eye-tracking data and for the automation of the evaluation process. The set of eye-tracking measures includes (using the definitions of Jacob and Karn (2003, p. 585)):

- *Number of fixations*: the number of fixations overall is thought to be negatively correlated with search efficiency.
- *Proportional time on each area of interest*: on each area of interest, this measure is thought to reflect the relative importance of the area.
- *Fixation duration*: mean fixation duration is thought to be related to difficulty with extracting information and with the depth of the required processing.
- *Number of fixations on each area of interest*: the number of fixations on a particular area (e.g., on an interface element) should reflect the importance of that area.

By using the measures, 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 data and relate them to the underlying processing or to the usability aspects are tasks yet not completely understood, as discussed by Goldberg and Wichansky (2003) and Jacob and Karn (2003).

Goldberg and Kotval (1999) also presented a classification of eye-movement measures. They proposed that a measure can be seen as temporal, if it describes a time-based property of a scanpath, that is, the time spent on a certain area of interest. For instance, the duration of a fixation is a temporal measure. Spatial measures, on the other hand, describe the spread and coverage of a scanpath. For instance, the number of saccades or scanpath length are both referred to as spatial measures. A transition matrix, according to Goldberg and Kotval (1998) is a spatial measure (and according to Goldberg and Kotval (1999) it is both a spatial and temporal measure) that contains the number of transitions of visual attention between areas of interest.

In their summary of recent usability studies, Jacob and Karn (2003) report on the most commonly used eye-tracking measures. Of these studies, 11 made use of *number of fixations*, seven reported the *proportional time* spent looking at each area of interest. Another six studies used *mean fixation duration* and *number of fixations* on each area, five employed the *mean gaze duration* and five reported the *rate of fixations per second.* This thesis, however, does not focus on providing an exhaustive list of all eye-tracking measures employed in usability studies, see (Jacob and Karn, 2003, Goldberg and Kotval, 1998) for a review and more details, respectively. It is worthwhile to note that, based on the data provided by Jacob and Karn (2003), an average of 14.6 participants take part in usability studies that incorporated eye-tracking and had been reported in a scientific forum.

There are, however, also eye-tracking measures that have appeared recently and have been employed to evaluate users' visual behavior. In particular, a shift can be observed from the measures computed from the raw eye-movement data to more sophisticated measures and procedures that take the specific context of a study into consideration. An example of context-specific eye-tracking measures can be found in the study of Yoon and Narayanan (2004). The authors refer to *coverage* as the percentage of areas of interest that were attended to for an interval greater than a predefined threshold. They also investigate the *order* of fixations to measure how systematically a user attends to casually related areas of interest.

While Yoon and Narayanan (2004) captured the correctness of the fixation sequence as a single number, also analyses of whole sequences of fixations – i.e. scanpaths – based on string editing seem to be increasingly popular. Some examples of this approach are the studies of Josephson and Holmes (2002) and Pan et al. (2004), which made use of string algorithms to the compare similarity of fifteen seconds long scanpaths of participants viewing repeating Web images and complete Web pages.

2.2 Interpreting Eye-Tracking Data in Usability Studies

Investigations that attempt to relate eye-tracking measures to aspects of usability belong to the retrospective use of eye-tracking. In the aforementioned summary of over twenty eye-tracking usability studies, Jacob and Karn (2003) acknowledge the early human performance investigation of Fitts et al. (1950) as the first usability study that incorporated eye-tracking. In the following we do not aim to review the long history of all the previous usability studies that have incorporated eye-tracking; instead, the aim is to highlight the typical approaches to eye-tracking data analysis and interpretation and to the research settings of the previous studies.

The often cited study of Goldberg and Kotval (1999) can be considered as following the experimental and reductionist tradition of HCI. The implicit hypothesis of the investigation – that is, whether there is an effect of poor or good design on user search strategies as described by the eye-tracking data – is tested using parametric statistical procedures. The resulting eye-tracking measures – taking the form of averaged data points at the end of the experiment – are supposed to provide information about the user strategies.

The experimental interface Goldberg and Kotval (1999) employed was artificially made and the task given to the participant consisted of only searching for targets with a focus on the speed. The duration of a unit of the resulting eye-tracking data averaged around 1.5 seconds.[3] However, the interaction with modern computer interfaces, such as with multimedia learning systems, can hardly be considered to be simple target search task without more elaborate goals.

Recent studies, such as the web-page usability evaluation of Cowen et al. (2002), recognize the problem and employ a more realistic task and context.[4] Finding no significant effects of task and page interaction on eye-tracking measures, the authors discuss the possibility that the whole strategy might comprise of more than one type of processing and it cannot be effectively evaluated by a single eye-tracking measure. Finding no significant effects using the quantitative statistical approach, the authors utilized the visualizations of the eye-tracking measures to find qualitative evidence for their discussion.

Cowen et al. (2002) argue for establishing benchmark measures and investigating the relation between underlying processing and eye-movement patterns. They also suggest that the visibility of a target might influence the patterns and that more tasks should be studied to uncover the factors affecting usability and eye-movement patterns.

Recently, Renshaw et al. (2004) explored the depth of analysis of eye-tracking

[3]The paper of Goldberg and Kotval (1999) reports averages of 2.17 and 2.53 fixations for good and poor interfaces, respectively.

[4]Mean total fixation duration in Cowen et al. (2002) ranges from 12 to 23.5 seconds.

measures and evaluated the design of two line charts. Total fixation duration, mean fixation duration, and fixation rate were examined for five areas of the charts, using a repeated-measures design and ANOVA. In addition, the temporal distribution of the measures was investigated by segmenting the whole session into four five-second intervals. This analysis revealed a persistent effect of the interval on the eye-tracking measures.

Although segmentation yields new insights into strategies during chart-reading, the interpretation of the results, as Renshaw et al. (2004) admit, is non-trivial. In addition, achieving the increased temporal resolution of the analysis cannot be at the moment fully automatized. Presumably, also in the study of Renshaw et al., the segments had to be hand-coded, an unwanted trade-off an eye-tracking researcher has to pay for the clearer view on the user strategies.

Although similar to previous investigations that employed relatively short tasks, the descriptive study of Josephson and Holmes (2002) stands out from the pool of studies reviewed in this chapter. To find representative sequences of fixations, Josephson and Holmes (2002) used a variety of sophisticated data analysis procedures, including string editing, multidimensional scaling, and hierarchical cluster analysis; no significance testing is reported. When facing the mixed results caused by between-subject variability and personal preferences, the authors turned to visually analysing and "eye-balling" the data sets to find explanations for the differences.

This overview has shown that eye-tracking studies have employed a variety of techniques to describe ocular behavior and relate it to usability and user strategies. The tasks given to participants were usually relatively simple, and the resulting sequences of eye-tracking data from which the measures were computed were limited to a range of seconds. This approach to data extraction, interpretation and its relation to underlying cognitive processing can be considered to belong to the reductionist tradition of human-computer interaction studies.

Yet, several eye-tracking researchers have converged on the finding that the analysis and interpretation of the resulting eye-tracking data is non-trivial, and both complicated and tedious (e.g., Salvucci and Anderson, 2001). Jacob and Karn (2003) even argue that "*[t]his daunting task remains a hindrance to more widespread inclusion of eye-tracking in usability studies*". In the following section, we make a distinction among three types of research situations and consider the "dauntingness" of the analysis of eye-tracking data and its relation with the complexity of the task and interaction.

2.3 Types of Research Situations

Modern computer applications – and generally all research situations where the stimulus is unpredictable and users can interact with the display system freely – introduce another challenge to researchers wishing to employ eye-tracking. The stimuli in many previous studies have been static and so have the corresponding areas of interest. Automation of the analysis of the resulting eye-tracking data is rather simple in these cases and most manufacturers support this task by their eye-tracking analysis software.

Let us consider, however, studies where the stimuli become dynamic. We can further distinguish between two types of such studies. In one class of eye-tracking studies the researcher has access to the system generating the display. This class includes, for example, studies of Web page browsing or studies of driving with a simulator.

The latter class of eye-tracking investigations includes studies in which the researcher does not have access to the system generating stimuli. Studies of driving on real roads or interaction with ambient environment belong to this class.

An access to the system calls that generate the dynamic content makes it possible to map the eye-tracking data on the content. Current eye-tracking data analysis tools, however, do not allow one to efficiently set up and perform the analysis of such data. To deal with the dynamic nature of stimuli, behavioral researchers, regardless of the type of the study, have often been left with only manual extraction and annotation of fixation data. The problem with this approach to analysis is that it requires considerable efforts on the part of the researcher, and perhaps is another hindrance to the widespread use of eye-tracking techniques in more complex and dynamic research situations.

Another consequence of dynamic scenes and free interaction in eye-tracking studies is that the resulting data might become sparse. To emulate real-world settings, researchers allow participants to perform the task voluntarily, without imposing a stricter control over what parts of the task the participants need to perform and how. This is the case in studies of programming behavior or in studies of driving, where in order to record natural behavior the participants need to be engaged in the task without restrictions. For example, a researcher might ask whether the output of a program has been visually attended more than some other representation of the program during a debugging task. It might turn out that only some of the participants willfully attend to the output of the program at all. This will cause the resulting sets of data to become sparse, and poses limits on parametric statistical procedures. Given the effort and expense related to conducting an eye-tracking study, the researcher might not choose to recruit more participants to ensure statistical validity of the sample. In addition, the sparseness of the data-sets might also be caused by technical problems in eye-tracking, which might render parts of the recordings unusable. Therefore, methods for dealing with the sparseness of the eye-tracking data need to be developed.

2.4 The Restricted Focus Viewer: An Alternative Tool to Track Visual Attention

An alternative method to eye-tracking, Focus Window Techniques (FWT) has been designed to reduce the technical problems inherent to eye-tracking. The FWT screen is blurred except for a small focused section that is supposed to be linked to users' foci of visual attention. If users wish to investigate another part of the display, they need to move the small focused section to that location. Because the technique cannot lose the gaze as eye-trackers sometimes do – because it is not based on the recording of the gaze direction from the eyes – it can be thought of as a partial solution to the sparseness problem.

Recently, the Restricted Focus Viewer (Jansen et al., 2003), a tool that implements FWT, has been introduced as an alternative tool to track visual attention. The tool blurs the computer display and restricts users to only a small focused spot within an otherwise blurred stimulus. When researchers employ the Restricted Focus Viewer (RFV) to track the visual attention, they need to prepare several versions of the stimuli with gradual levels of blurring. As a participant moves the focused region using a computer mouse, the RFV records the movements of the spot over the stimuli and stores them for later analysis. The tool collects the timestamped data for the mouse and the keyboard events, the index of the focused region, the total durations of sessions, and other events. Voice protocols can be recorded along with the interaction data. The RFV tool, naturally, is not capable of collecting visual attention data when the blurring is turned off.

The Restricted Focus Viewer has been validated in two experiments run by Jansen et al. (2003); however, these validations involved only a relatively simple reasoning tasks with visual stimuli. In other studies, RFV-based technology has been applied 1) to discover the strategies of participants debugging computer programs with an aid of multiple and linked visual representations of the programs (Romero et al., 2002a,b, 2003a,b), 2) to investigate the usability of hyper-linked documents (Tarasewich and Fillion, 2004, Tarasewich et al., 2005), or 3) to research shifts of visual attention during the integration of text and graphics (Futrelle and Rumshisky, 2001). Similar to eye-tracking studies, the analysis and interpretation of the data recorded by RFV and their relation to the investigated task is up to the researcher. Usually, the so-called areas of interest (AOI) are defined within the interface and several metrics are computed, such as the total or proportional time spent on an AOI. How RFV-based measurements of visual attention during programming compare to those of eye-tracking, is an open question.

2.5 Summary

Attempts to solve the major methodological challenges in eye-tracking, that is the problems of how to automate the analysis of eye-tracking data and how to relate the resulting measures to the underlying processing, have not yet arrived at a coherent understanding of the process and the relation itself. Previous eye-tracking studies have mostly taken the reductionist-controlled approach to data collection and analysis. While this approach might work where the research situation is relatively controlled and simplistic, it is not clear whether it also applies to more complex and natural domains. This gap necessitates an increase in the knowledge available about the connection itself in a specific domain and the knowledge about how to conduct the analysis to investigate the link in general.

Considering the dynamic modern computer interfaces, more studies are needed to investigate the properties of eye-tracking data in that type of interaction. While the technical problems with eye-trackers seem to be progressively eliminated by industrial vendors or by alternative research tools to track visual attention, methodological problems still persist. The lack of methods to analyze and interpret the eye-tracking data together with the low efficiency and limited functionality of available analysis packages continue to inhibit the spread of the promising technology.

Chapter 3

Behavioral and Cognitive Aspects of Programming

PROGRAMMING has long been recognized as a challenging and cognitively demanding task (Shneiderman, 1986, Hoc et al., 1990, Détienne, 2002). The complexity of programming is multifarious, and lies in the need to acquire the understanding of and to coordinate many entities. These elements are often hidden, latent, interconnected and interdependent. Success in programming requires specialized knowledge and expertise in a variety of subtasks and strategies. Some even claim that "*programming is far more complex than usual human mental activities studied by psychologists*" (Weinberg and Shulman, 1974, p.70).

One of the central skills in many programming tasks, such as during the creation, maintenance and modification of a software product, is program comprehension – the ability to understand programs written by others (Littman et al., 1986, Pennington, 1987), and also one's own programs after an extended period.

Comprehension of a program means constructing an internal representation (mental model) of the target software (von Mayrhauser and Vans, 1995). That means that while comprehending a program, a programmer has to build an understanding of what the program does and how it does it. Elements of a mental model include, for example, text structures, chunks, plans, hypotheses, beacons and rules. During program comprehension, a programmer also needs to uncover the various implicit relations and hidden dependencies in the program and integrate those with the mental representation of the program. The correct comprehension of a program, in turn, allows the programmer to debug and modify the program. Thus, program comprehension is a complex cognitive process that involves several strategies, extensive application, domain and program knowledge, and the coordination and integration of new and existing knowledge.

Comprehension, a critical cognitive activity in programming (Brooks, 1983), is, at the same time, one of the major activities carried out during software development. Many see program comprehension as central for successful maintenance (e.g., Littman et al., 1986, von Mayrhauser and Vans, 1995, Corritore and Wiendenbeck, 2001). Rajlich (1994) claims that during software maintenance the comprehension related costs make up 50% of total lifecycle costs. Layzell et al. (1993) estimate that the maintenance and support costs make up more than half of the total costs. Therefore, studying and understanding how a programmer approaches comprehension is important for supporting both task completion and productivity. In order to develop tools to aid programmers in comprehension, the very process of program comprehension needs to be first understood.

Several models of program comprehension have been proposed in the previous research. In the following we will first review the related research in program comprehension, and then we give attention to the methodologies employed to study the underlying cognitive processing.

3.1 Program Comprehension, Debugging, and Skill Differences

Theories of program comprehension, the models of code cognition, attempt to explain how a programmer approaches and understands the program code. While all comprehension theories agree that a programmer uses existing knowledge during comprehension, they disagree about how the comprehension processes are carried out and coupled with the knowledge to obtain the new knowledge about the program. In broad terms, three types of program comprehension models were established and investigated in past research: *top-down* models initiated by the work of Brooks (1983) or Letovsky (1986), *bottom-up* models investigated, for example, by Pennington (1987), and *as-needed* models proposed by Littman et al. (1986). For a survey of the research about comprehension before 1995 see (von Mayrhauser and Vans, 1995). The research of program comprehension is still flourishing, as evidenced by the models proposed by Mayrhauser and Vans (the integrated model of code cognition) (von Mayrhauser and Vans, 1996) or O'Brian et al.'s recent findings (O'Brian et al., 2004).

How a novice becomes an expert is an important question in studies of problem solving and in studies of psychology of programming. Numerous studies have investigated the effects of expertise on the performance and strategies of programmers. This research has clearly shown that expert programmers outperform novices in several aspects. Expert programmers are superior in terms of domain knowledge and strategies that lead to improved performance. In previous studies, experienced programmers have found more bugs, found them faster, and tended to spend more time on building a mental model of the problem (Gugerty and Olson, 1986b). Other studies have shown that experts are also more able to remember specific parts of the source code (Fix et al., 1993), they focus on relevant information needed to solve the problem (Koenemann and Robertson, 1991), they are not committed to one interpretation as novices (Vessey, 1985), and are, therefore, able to change their strategies as needed.

Despite the superiority of expert programmers in terms of performance and knowledge, several studies converge on the finding that the strategies involved in comprehension do not differ between novice and expert programmers. Brooks (1983) suggested that differences in comprehension strategies caused by expertise might not be as large as those caused by differences in programming and domain knowledge. Gugerty and Olson (1986a) have shown that while experts outperform novices in debugging, the proportional times spent on different activities during comprehension are similar for the two groups. Based on the similarities found in the strategies, it has been suggested that experts can make better use of the information and the knowledge available.

Burkhardt et al. (2002) studied the effects of expertise and task on comprehension of an object-oriented program. It has been found that experts are better in constructing mental representations. In particular, when asked to comprehend a program for the purpose of documentation, experts constructed different situation models than novices, although the program models were similar. The number of comments produced by experts and novices was not different; however the source of the comments and the expertise interacted significantly. When asked to comprehend a program for the purpose of reuse, no effects of expertise on performance was found. Furthermore, strategies exhibited by novice and experts, as measured by reuse activities, did not differ.

In summary, the previous findings seem to indicate that the increased performance in program comprehension or debugging is not accompanied by a difference in the externalized strategies; instead, larger knowledge base, better mappings between the program domain and programmer's cognitive models and resources, and experience in a familiar application domain all enable expert programmers to have superior performance.

3.2 Investigating The Behavioral and Cognitive Aspects of Programming

Given the importance of program comprehension and its understanding, there is a great need for research methodologies that allow for a proper and effective investigation and analysis of the underlying processes. How do researchers then study programmers performing a programming task like comprehension?

Several reviews of the methodologies employed in studies of programming have been published. For example, a broad overview of the current research methods used in the psychology of programming can be found in a classic collection edited by Hoc et al. (1990). Although Shneiderman (1986) admits that there are many ways to do research on programmers, he suggests that controlled experiments are the authoritative methodology to produce reliable results.

The experimental tradition in the studies of programming is indeed notoriously famous. Many empirical studies of programming borrow their methodologies from experimental psychology (Blackwell et al., 2001); Blackwell et al. (2001) argue that hypothesis testing is the main research technique employed in the studies of programming. In the larger domain of empirical software engineering, utilizing hypothesis testing and the associated statistical significance testing has "*become the backbone of the topic*" (Miller, 2004, p. 183). Miller (2004), however, warns against the problems with application of the technique. For example, he points out, human participants in software engineering experiments have large variations in ability.

Individual variability among programmers is noted by many sources as discussed by (Sheil, 1981). Regarding the individual variability among programmers of a similar background, Sheil (1981, p.702) points out that in experimental designs "*slight systematic differences between conditions tend to be washed out by large within-condition variation.*" Yet, studies in psychology of programming tend to employ controlled experimentation to study human information processing and problem-solving.

For instance, Gilmore (1990) presents the studies of programming as having a historical tendency toward experimentation and hypothesis testing; however, he also identifies a shift towards observational and exploratory data collection. In addition to hypothesis testing, observation indeed seems to be the second most popular type of methodology employed in the studies of programmers, as seen from surveys of program comprehension studies and data collection techniques. Mayrhauser and Vans present an overview of the main program comprehension studies conducted prior to 1996 and their experimental settings (von Mayrhauser and Vans, 1996). A recent paper of Lethbridge et al. (2005) surveys the data collection methods employed in software engineering research. In all of the surveyed studies, however, no attempt to investigate visual attention and its role during comprehension had been reported.

Regarding the data collection method employed to capture and analyze the underlying processing, it turns out that most of the studies in the field made use of some version of the think-aloud method (e.g., Brooks, 1977, Vessey, 1985, Letovsky, 1986, Littman et al., 1986, Pennington, 1987, von Mayrhauser and Vans, 1996, Vessey, 1985, Ko and Myers, 2004, Owen et al., 2006, and many others). Using on the think-aloud method, the studies typically conducted some version of protocol-analysis (Ericsson and Simon, 1984) to explore the underlying cognitive processing. For instance, Littman et al. (1986) conducted an empirical study of ten professional programmers performing a maintenance task. Using think-aloud protocols they showed that there were two distinct comprehension strategies, a systematic and an as-needed strategy. Programmers using the systematic strategy were able to acquire the knowledge about the program and to perform a modification task successfully.

The approaches based on verbal reports, however, have been criticized widely (e.g., Branch, 2000, Nielsen et al., 2002, van den Haak et al., 2003). While the think-aloud protocol is undoubtedly a useful and well-grounded tool to study strategies during programming, it does not provide detailed information about what particular sources of information programmers attend to as they carry out the task. Analysis of think-aloud data is also difficult and time-consuming (Chi, 1997). Some attempts to formalize and improve the methodology of protocol analysis, however, exist (Chi, 1997, von Mayrhauser and Lang, 1999).

When verbal reports are applied to study novice participants, it might be argued that the need for verbalizing thoughts might interfere with otherwise natural processing by adding an extraneous cognitive load (Nielsen et al., 2002). Because many of the studies of programming strategies are conducted with novice participants performing some complex task of programming, it might be hypothesized that the tasks impose a heavy cognitive load. This extraneous load might result in an interference with their normal problem-solving strategies and, therefore, bias results. In the context of usability studies, Goldberg and Wichansky (2003) also point out that individuals are not aware of some aspects of behavior, such as the focus of attention during a task, and these are, thus, hard to verbalize.

Besides the tools and techniques for recording the human behavior during programming, also methodologies to analyze the cognitive processing and the resulting outcomes are needed. The outcomes of program comprehension are the mental models acquired by the programmer during the comprehension task. Good and Brna (2004) developed a schema based on the types of information found in comprehension summaries. The schema allows a researcher to evaluate the mental models acquired during comprehension in terms of the proportions of different information types, such as statements related to control-flow, data-flow, function or operation.

In addition, it allows a researcher to classify the statements according to the level of abstraction describing the program objects. Recently, Byckling et al. (2004) suggested that the *irrelevant* category should also be included in the schema.

Another alternative option to investigate the mental models are comprehension questions, a method originally developed by Pennington (1987). The application of comprehension questions is straightforward. After a comprehension task, or a part of it has been completed, the participants are asked several questions that test the information that they have available.

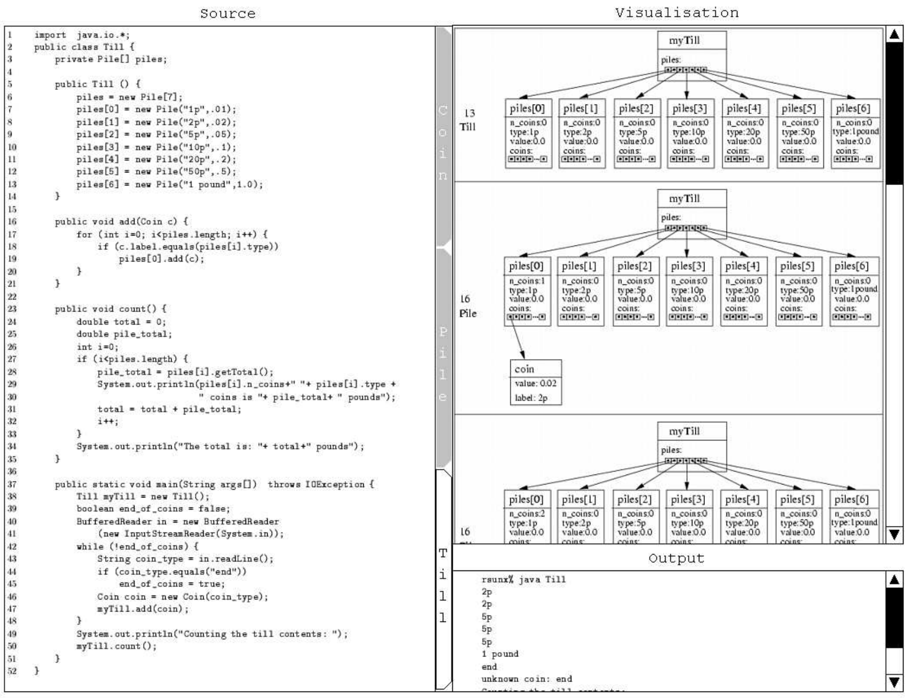
Comprehension questions, questionnaires, or comprehension summary analysis answer questions about the outcome of the process, but not about the process itself. Some researchers have therefore made use of a combination of two or more methodologies, such as Burkhardt et al. (2002) who used both comprehension questions and think-aloud protocols. Both the process and the outcome of program comprehension can then be studied using a combination of two methods. A relatively new data collection technique employed in the studies of the processes involved in programming is visual attention recording.

3.3 Studies of Visual Attention during Programming

Analysis of behavior based on visual attention data is becoming increasingly popular in studies of programming. Previous studies in programming that employed visual attention tracking focused, for example, on how programmers read the source code (Crosby and Stelovsky, 1990), how programmers make use of and coordinate multiple representations (Romero et al., 2002a,b, 2003a,b) or on the effects a graphical visualization of a program has on the visual attention patterns of novice programmers (Nevalainen and Sajaniemi, 2005).

Crosby and Stelovsky (1990) employed an eye-tracker to explore the patterns of programmers' visual attention while reading a binary search algorithm written in Pascal. Using mostly qualitative approaches, but also parametric tests, the authors analyzed fixation times and the number of fixations to reveal the strategies involved in reading source code. Crosby and Stelovsky argue that while the participants with greater experience paid attention to meaningful areas of source code and to complex statements, novice participants, on the other hand, visually attended to comments and comparisons. Both groups paid the least attention to the keywords, but did not exhibit any systematic differences in reading strategies. Crosby and Stelovsky also reported that the two most similar scanning patterns while reading an algorithm belonged to subjects from opposite experience groups.

The attempt of Crosby and Stelovsky (1990) might be considered to be pioneering; however, only one representation of program was used at the time. Also, their research focused mainly on the critical, but surface features of code and not on the multiple and dynamic representations often present in modern program development interfaces.



```
import  java.io.*;
public class Till {
    private Pile[] piles;

    public Till () {
        piles = new Pile[7];
        piles[0] = new Pile("1p",.01);
        piles[1] = new Pile("2p",.02);
        piles[2] = new Pile("5p",.05);
        piles[3] = new Pile("10p",.1);
        piles[4] = new Pile("20p",.2);
        piles[5] = new Pile("50p",.5);
        piles[6] = new Pile("1 pound",1.0);
    }

    public void add(Coin c) {
        for (int i=0; i<piles.length; i++) {
            if (c.label.equals(piles[i].type))
                piles[0].add(c);
        }
    }

    public void count() {
        double total = 0;
        double pile_total;
        int i=0;
        if (i<piles.length) {
            pile_total = piles[i].getTotal();
            System.out.println(piles[i].n_coins+" "+ piles[i].type +
                               " coins is "+ pile_total+ " pounds");
            total = total + pile_total;
            i++;
        }
        System.out.println("The total is: "+ total+" pounds");
    }

    public static void main(String args[])  throws IOException {
        Till myTill = new Till();
        boolean end_of_coins = false;
        BufferedReader in = new BufferedReader
            (new InputStreamReader(System.in));
        while (!end_of_coins) {
            String coin_type = in.readLine();
            if (coin_type.equals("end"))
                end_of_coins = true;
            Coin coin = new Coin(coin_type);
            myTill.add(coin);
        }
        System.out.println("Counting the till contents: ");
        myTill.count();
    }
}
```

```
rsunx% java Till
2p
2p
5p
5p
5p
1 pound
end
unknown coin: end
```

Figure 3.1: An IDE using multiple representations to present a program.

Multiple representations are commonly found in modern programming environments, where the main representation (e.g., the code) is accompanied by another view on some aspect of the program. Figure 3.1 shows an example of a multirepresentational programming environment presenting the source code of a program, a graphical representation of it, and its current output.

One of the first attempts to investigate visual attention patterns when debugging in multirepresentational environments was that of Romero et al. (2002b). The authors investigated the coordination strategies during debugging by measuring the overall long-term times spent on different representations of a program (Romero et al., 2002b). Using an environment based on RFV technology (Jansen et al., 2003) and the traditional hypothesis testing framework and parametric statistical procedures, Romero et al. provided insights into representation use and visual attention behavior during debugging.

The measures of visual attention were computed by taking the same long-term-aggregated perspective as the eye-tracking measures in the usability studies introduced above; a measurement in this particular study was an averaged aggregate of all data between the beginning and the end of a debugging session. Programming experience, according to Romero et al., promoted more balanced switching behavior between the adjacent representations of the program.

Also, other studies of visual attention in programming tended to analyze the data sets on the most aggregated level (e.g., Nevalainen and Sajaniemi, 2005, 2006). To investigate the link between the underlying processing and overt visual attention patterns, in these, and many other studies, the hypothesis testing framework had been utilized. Similarly as in other investigations, behavioral data were approached from the long-term-means perspective.

For example, Nevalainen and Sajaniemi (2005) presented twelve participants with a program comprehension task and two programming tools – a traditional textual environment and a graphical visualization system – and compared the resulting visual attention patterns. As the dependent eye-tracking measures, the authors used absolute viewing times (the sum of all fixations during each of the comprehension sessions) and the proportions of these times over three discrete areas. Differences were found in the ways participants targeted their visual attention while working with a graphical program visualization tool compared to the patterns exhibited while working with the textual programming tool. It seems likely, however, that the differences in the absolute viewing times might have been related to or biased by the time needed to complete the task using each of the tools. Interestingly, using any of the tools produced about 58-64% of time being spent looking outside code or visualization. In addition, no significant effect of a tool on the mental models created was found.

In another program comprehension study, Nevalainen and Sajaniemi (2006) employed a between-subject design and compared the total proportional fixation times on the elements of a display between two groups using two different versions of a program animation tool. Similarly to previous studies, the eye-tracking measures were represented by long-term (approximately 15 minutes) averages. These data sets were then analyzed using parametric statistical procedures. They found out that most of the time was spent on reading the code, regardless of the group the participants were assigned to. To investigate the results in a finer detail, Nevalainen and Sajaniemi reported employing also a qualitative analysis of short segments of video protocols with a superimposed gaze-path.

The approach of Uwano et al. (2006) for analyzing visual attention patterns during programming is a contrast to the mainly quantitative methods of the studies introduced above. Uwano et al. (2006) studied five intermediate programmers during six short source code review tasks as they attempted to find a bug. Instead of evaluating the recorded visual attention using statistical procedures, the authors made use of the visualization of the line of code that a single programmer was attending. This vertical point was plotted against time and the visualization served then as the source of further analysis; this included, for example, estimating the time a certain proportion of source code had already been attended.

The findings of Uwano et al. related to retrospective thinking suggest to use the recorded eye-tracking patterns for educational or training purposes. These results can be seen as supporting the motivation of Stein and Brennan (2004). Stein and Brennan made use of gaze recordings of expert programmers during debugging to support the performance of other programmers. Another group of programmers found bugs more quickly after the gaze of one of the expert programmers had been replayed to them.

Little has been done in the past to investigate how eye-tracking data can be utilized to evaluate user strategies and the underlying processing during programming tasks with dynamic computer environments. Although novel in employing multiple representations of a program (e.g., Romero et al., 2002b) or studying short-term effects of program animation on visual attention (e.g., Nevalainen and Sajaniemi, 2005), researchers have often approached the analysis of eye-tracking data in programming only from a long-term, global point of view. The data have been treated as aggregate means over a whole comprehension session. A single eye-tracking measure, such as the mean fixation duration or the total fixation time, is employed to describe or evaluate a strategy of a number of participants performing a rather complex cognitive process, such as comprehending a computer program.

Studies of visual attention in programming tend to be quantitative and often reduce the situation to a number of eye-tracking measures. These approaches might be an oversimplification of the underlying processes and demonstrate a need to develop a more thorough understanding of the strategies involved in programming. To characterize and analyze the cognitive processes involved in programming in greater detail, therefore, other approaches need to be involved.

Chapter 4

Improving Methods to Analyze Visual Attention Strategies in Studies of Programming

"Aut Viam Inveniam Aut Faciam."
– Hannibal Barca

IN the following sections, the main findings from the original reports are presented with respect to the main focus of this thesis. The following sections consist of a brief summary of the main objectives of each of the reported studies, and a summary of the findings as reported in the papers. In addition, a discussion of the findings and the methodological issues with respect to the thesis' main aims is presented for each publication. For the detailed results and exact tabulated values, we refer the readers to the publications attached at the end of this thesis.

At the beginning of the research process, we adopted three eye-tracking measures (the mean fixation duration, the proportional fixation time, and the frequency of switching between areas of interest) and quantitative analysis methods to investigate visual attention patterns of computer programmers.

For the first measure, the data were usually aggregated and analyzed with regard to an area of interest or in overall for the whole display (e.g. a mean fixation duration on output, and an overall mean fixation duration). For the proportional fixation time, or the proportional count of fixations, the resulting data were calculated and analyzed for each area of interest. Finally, for the frequency of switching, the data were analyzed with regards to either the type of switch (e.g. any switch between code and output) or/and for overall switching frequency regardless of the origin and destination.

4.1 P1: A replication of a previous study in programming using visual attention tracking

4.1.1 Background and aims

In (**P1**) we compared two tools to track visual attention during programming. We conducted a full replication of a previous study of debugging that employed a tool based on the FWT to track visual attention of programmers. In addition to tracking visual attention using the RFV, we recorded the visual attention of programmers also by a remote eye-tracking device. To investigate the possible effects of the RFV tool on the strategies of programmers, participants performed one debugging task with the restrictive view and another task without the restrictive view. The analysis of the resulting data consisted of performing comparisons and correlations of the three streams of visual attention data: a stream of data acquired by the RFV under the blurring condition, a stream of data acquired by an eye-tracker under the blurring condition, and a stream of data acquired by an eye-tracker under the unrestricted full-display condition.

4.1.2 Results

An effect of blurring on the frequency of visual attention switches per minute and an effect of tool on the frequency of switching were found. Under the blurred condition, the programmers performed less switches than with the unrestricted display and the eye-tracker measured more switches than the RFV tool. In addition, we discovered an interaction effect of the tool and the type of switch on the number of switches. The switching frequency data from RFV were not correlated with the data from the eye-tracker.

Finding the effects of the RFV tool and blurring on the visual attention data, we investigated whether and how the blurring might have affected on the debugging strategies and performance of programmers. These behaviors were measured only using the eye-tracker. While the performance of more experienced programmers slightly increased when the display was blurred, there was no effect of the blurring condition on the overall performance of programmers.

The proportional time spent on either one of the three main representations was not affected by the blurring of the display. It is also worthwhile to note that the distributions of the times spent on the main areas of the display were statistically not different for the less and more experienced programmers, suggesting that both groups engaged in similar strategies; however, an observation of the graphs of the proportions indicated that more experienced programmers tended to attend the code slightly more than the less experienced programmers.

Analysis of the frequency of visual attention switching indicated an effect of blurring on the strategies of programmers; more experienced programmers performed significantly fewer switches when the display was blurred. A correlation of the frequencies of switching between the two conditions showed that more the experienced programmers might have changed their switching strategies due to the blurring. While not explicitly stated in (**P1**), the switching frequency of the two groups of programmers was similar when the display was not blurred. This observation also suggests that the dynamics of the coordination strategies was similar for the two groups.

Finally, the mean fixation duration was measured and analyzed. This analysis showed a constant increase in the mean fixation duration caused by the blurring, and shorter mean fixation durations of more experienced programmers. Less experienced programmers did not exhibit any significant differences in the mean fixation durations between the two conditions. Interestingly, there also was an effect of the representation on the mean fixation duration; a graphical representation was attended to using the shortest fixations.

4.1.3 Discussion of the findings

Although the RFV can solve the technical problems inherent in eye-tracking technology – such as noise in the data stream or the loss of gaze – it does produce essentially the same stream of complex data as the eye-movement recording does; and so it does not simplify the analysis and interpretation of data. On the grounds of the results, we have to warn the potential users of the RFV against employing the tool to track visual attention of users performing complex problem-solving tasks. In our study of debugging, eye-tracking clearly outperformed the RFV in terms of the validity of the visual attention data measurement. At the same time, the RFV technique was shown to interfere with strategies of expert programmers and caused higher mean fixation duration.

4.1.4 Methodological discussion

This study, from the methodological point of view, made use of the hypothesis testing framework and took a long-term perspective on eye-tracking measures. A single session was limited to ten minutes and also the corresponding three eye-tracking measures were calculated to represent the strategy of the programmers during the interval. No significant differences in the strategies under the natural condition were found in terms of proportional fixation time and frequency of switching between areas of interest. Yet, when video replays were analyzed, some low-level differences in the strategies of the programmers were found. These could not have been described using the employed approach.

The integrated development environment employed in this study was artificial. Although participants could select which class of the source code would be shown and what part of visualization would be displayed, the content of each was static and the environment did not allow any other interaction. In the following study we, therefore, employed a tool that allowed for a more natural interaction and we employed only eye-tracking because of the problems found with the FWT-based technique.

4.2 P2: An eye-tracking experiment in a dynamic program visualization environment

4.2.1 Background and aims

In the second paper (**P2**), we began to evaluate the visual strategies of programmers during the animation of a program. The study was designed to determine, whether there is an effect of experience on the visual attention patterns of programmers. Similar to the previous study, we employed a remote eye-tracker and the three eye-tracking measures to capture and describe the patterns of visual attention.

There are, however, several differences between the first and the present study. First, the task given to participants differentiates the studies; in the present study participants were asked to comprehend a program to write a summary of it. Second, the environment employed in the study allowed for dynamic visualization of program execution; participants could invoke the visualization as often as they found it necessary. Third, participants could voluntarily select whether, and how, to use the visualization of the program or not. As a consequence of these two differences, the whole comprehension session comprised of phases with and without animation being displayed on the screen and these phases could occur irregularly during the session. Finally, the participants were not limited in the time allocated to conduct the task.

Except for one alternation, the resulting data were analyzed using a similar approach as in the first study: the whole screen was divided into areas of interest (code, visualization, output, and control), eye-tracking measures were aggregated in time and across the areas, we used parametric quantitative statistics, and visualized the data as plots of measures over the areas of interest. The difference in the analysis approach was that we included only the eye-tracking data during the animation of a program, as that was the only time when all representations were concurrently available. We selected this approach to segmentation to avoid the contamination of the results by alloying eye-tracking data from the two different tasks: multiple representations coordination and code-reading.

4.2.2 Results

More experienced programmers performed the comprehension significantly faster than less experienced programmers and spent proportionally more time reading the source code. Mean fixation duration of experts was significantly shorter than that of less experienced participants on all but the control-area. The effect of area on the mean fixation duration was found to be significant at $p = .052$. From the areas that contained any information related to the program, the shortest mean fixation durations were on the output of the program, followed by the fixations on visualization, and the longest fixations were paid on the source code of the program.

However, other two eye-tracking measures, relative fixation count and switching frequency, appeared not to be sensitive to the difference in programming experience. There was an effect of area of interest (representation) on the proportion of fixations. Over a half of the time of the animation, all programmers visually attended to the visualization (about 55% of all fixations); second mostly attended to was the textual representation of the program (about 40% of all fixations). The raw difference of approximately 15% between the proportional times spent on these two representations was statistically significant.

Programmers of both expertise groups performed most of the visual attention switches between code and visualization. Overall, less experienced participants tended to switch slightly more.[1]

4.2.3 Discussion of the findings

Consistent with the findings from the previous study (**P1**), mean fixation duration was found to be sensitive to the level of expertise. The findings related to completion time are also consistent with previous studies of programming. However, the expected differences in visual strategies did not materialize.

To our knowledge, (**P2**) had been the first study to employ eye-tracking to examine differences in visual strategies during program visualization. Finding differences in the time to carry out the task, in the proportional time spent animating the program, and in the mean fixation durations – but not in the other eye-tracking metrics – this study sets the goal for the following investigations: Understanding

[1]The original paper fails to report that a t-test on the overall switching frequency showed no significant difference between the groups ($t(14) = .55$, *ns*).

how to better interpret eye-tracking data, and how to identify the visual attention strategies using eye-tracking and more efficient analysis methods and more accurate measures.

4.2.4 Methodological discussion

There are several possible explanations for the surprising findings of this study. While the discussion in (**P2**) considers confounds related to experimental setting, such as sample size or a lack of difference in expertise, it is also possible that there is, indeed, no difference in visual strategies.

In hindsight, it becomes, however, clear that the analysis approach employed in the first two studies was not adequate to describe the eventual difference in the visual strategies. The quantitative approach conducted on the relatively long segments of data and with large areas of interest cannot completely describe all the nuances of programmers' visual behavior. It is thus possible that the areas of interest were designed too coarsely and the actual differences happened on a finer level of detail.

4.3 P3: Effects of expertise on visual attention patterns during dynamic program animation: a detailed analysis

4.3.1 Background and aims

Although the previous study managed to describe some aspects of visual behavior during programming, it indicated that the actual differences in visual strategies might have happened on a level that was not captured by the selected analysis approach. Therefore, in the third paper (**P3**), we focused further on the effects of expertise on visual attention patterns as measured by an eye-tracker.

To achieve a more detailed level of analysis, we further divided the interface of the programming tool into smaller areas of interest. In particular, the visualization of a program contained views of different aspects of program execution. Thus, the visualization area was further divided into finer level areas matching the different views. In total, there were seven areas of interest. As a consequence, twenty-one different types of switches were possible to perform while changing focus of attention from one area to another. This allowed for comparing and contrasting the granularity of the present analysis to the one conducted in the previous paper.

Another new level of detail of analysis introduced in this study concerns the evaluation of the impacts of different programs on visual behavior during comprehension. Finally, we also conducted the analyses of 1) the correctness of program summaries using a grade, and 2) the resulting mental models of programmers using the approach of Good and Brna (2004).

4.3.2 Results

No significant effects of experience on performance and on the contents of mental models were found. We also found no significant effects of the target program on these dependent variables. However, there was an interaction effect between the program and the type of the information in the resulting mental models.

Proportions of fixation counts were affected neither by expertise nor by program; however, area of interest had an effect on the distribution of fixations. Also, there was an interaction of program and area of interest on the proportional fixation count.[2] Other interactions were found to be not significant.

Frequency of visual attention switching was not significantly affected by the program, while there was an effect of experience approaching significance at $p =$ .089. The type of switch had a significant effect on the switching frequency. The interaction of program and switch type on the switching frequency was significant.[3]

The effect of experience and the interaction effect of experience and area on the mean fixation duration were significant. There was no effect of program and no interaction effect of program and area on the mean fixation duration. The mean fixation duration of more experienced programmers was consistently shorter than that of less experienced programmers, with the exception of the control area that did not show any information related to the program and the measure was approximately equal for both groups.

4.3.3 Discussion of the findings

This detailed analysis uncovered new facts about visual attention during interaction with multiple representations. First, it showed that there were interaction effects of

[2]The paper misses to report on a further analysis of this interaction effect. A series of adjusted pairwise t-tests discovered no statistical differences in proportional fixation counts between the programs on code, output, control, and constant areas; however the measure was significantly greater on method area during program 2 ($t(16) = 3.79$, $p = .002$), significantly smaller on expression area during program 2 ($t(16) = 2.25$, $p = .039$), and significantly smaller on instance area during program 2 ($t(16) = 4.64$, $p < .001$).

[3]Pairwise comparisons, not shown in the paper, revealed that the frequency of switches was: 1) higher between code and method areas in program 2 ($t(15) = 2.97$, $p = .009$), 2) lower between code and instance areas in program 2 ($t(15) = 3.48$, $p = .003$), and 3) lower between expression and instance areas in program 2 ($t(15) = 6.33$, $p < .001$).

program on visual attention strategies during animation. Attention to different areas changed when the program changed and also the type of switching followed these differences. This finding can be explained by the differences in the programs. When the programs differed, their resulting visualizations resulted in the graphical representations that differed in many aspects. For example, when the program contained more method calls – such as the recursive binary search program – consequently, the visualization showed more information related to method calls. The increase in the number of fixations that landed on the area that displays the visualization of the method calls seem to be a natural consequence, as well as is the increased number of switches per minute between the code and that area.

Although a great deal of variance was found in the strategies, the visualization of the measures shows that more experienced programmers focused more on code than the less experienced programmers did. The output and control areas were attended with minimal number of fixations and we can safely discard their influence on the distribution of fixations on other areas; therefore, the increase of the visual attention on the code resulted in a lower number of fixations the more experienced programmers paid to the visualization than the less experienced did. The increased granularity of the areas of interest revealed that proportional fixation count is not, in a statistical sense, sensitive to the effect of expertise. However, it allowed for delineation of the distribution of visual attention on semantically distinct areas of visualization and switching between them during program visualization.

The findings related to the frequency of visual attention switching seem to be in partial contrast with those related to proportional fixation count. The effect of experience on the switching frequency had not been, strictly speaking, statistically significant. However, it approached what is traditionally recognized as an acceptable level of statistical significance. It is hard to predict, however, whether if more participants would have taken part in the study, would the analysis of variance have shown a significant difference. Further studies need to be conducted to confirm or reject that eventuality. Regarding the effect of program, the programmers were switching their attention at about same rate; however, the programs influenced the type of the switching.

In general, the strategical difference can be described on two levels: overall, both groups spent about same time on different representations of the program and switched between code and visualization at approximately the same rate; the more experienced group switched the visual attention less often than the less experienced group, a difference obvious especially within the four areas of visualization. Once the more experienced programmers targeted their attention on the visualization, they performed less switches within that area. Out of these switches, only the difference in switching frequency between instances and method areas turned out to be significant; however, because a great number of comparisons were performed, the significant outcome of the test could have also occurred by chance.

Analysis of mean fixation durations confirmed the findings from the previous studies. Increased experience in programming is characterized by shorter mean fixation duration on program-related areas and in total. In addition, the effect of area on mean fixation duration shows that information visualized in expressions required the longest processing, followed by the mean fixation duration on code, instances area, method area, in that order. The output of the programs and constant area were attended with the shortest mean fixation durations.

4.3.4 Methodological discussion

From the methodological point of view, the results allow for a comparison of the granularity of areas of interest as presented in (**P2**) with the increased spatial granularity of the present study. The increase in the granularity of the analysis – that is, when a larger area of visualization is split into several smaller but semantically distinct areas – brings about several interesting observations that have implications for the use of this approach to study programming.

When the number of areas of interest increased, the analysis of frequency of switching began to provide us with new perspectives on some differences in how groups of programmers of varying experience deploy their visual attention during comprehension. While programmers spent about same time visually attending the regions of the interface, the temporal allocation of fixations between these regions seems to better distinguish skilled and novice programmers.

Therefore, further analyses of strategical differences in visual attention during programming with multiple representations should concentrate on the *temporal properties* of the strategies as seen in eye-tracking data collected from *semantically meaningful* areas of interest. A switch of visual attention between two areas, compared to the proportional gaze time on the two areas of interest, could become a promising dynamic eye-tracking measure of cognitive activity.

There are, however, also drawbacks to using a fine-level of detail in the design of areas of interest. Using the statistical approaches, the high number of areas might result in a higher number of tests for post-hoc comparisons. To achieve a reasonable power of statistical comparisons, a high number of participants have to be involved in the experiments or the overall significance level has to be adjusted.

Similar to the previous report, it seems that the variability in visual strategies impairs the ability of the analysis method to reliably detect the differences. When the number of areas increases, some participants might choose not to attend to particular areas. This behavior results in empty cells in the resulting data set – the data set becomes sparse – and, thus, it poses problems for the statistical analysis. Due to the missing values, whole cases of data are excluded from the data-sets, further decreasing the power of the statistical tests.

One way to circumvent the problems is through the use of visualization of fixations superimposed over the interface. (**P3**) shows a static visualization of the most attended areas. This visualization, however, provides only a static view on the strategies. Video recordings of gaze path of individual participants were instrumental in driving the subsequent research, because the more subtle differences in visual strategies could be observed (regardless the non-significant findings of quantitative statistical methods).

Another challenge of using a finer-level granularity in the analysis of the eye-tracking data is the amount of labor associated with preparing the data sets for analysis. This is especially true for segmenting the eye-tracking recordings into shorter events, therefore, increasing the number of analysis units. Current analysis software packages do not allow handling this task effectively; the duration of the analysis intervals varies between subjects so that an alignment of the respective intervals in time is not possible to do with the current tools.

The present findings do not present enough evidence to completely describe the visual attention strategies of programmers. Instead, they provide a static and relatively coarse view on the use of multiple program representations. It is likely that programmers do engage in a variety of strategies and, thus, they do change their strategies during the process, and they make use of different information at different stages of the comprehension. In other words, the results suggest that it is likely that the roles of the representations change during different stages of building a mental model of a program.

Whether the building of the model and dynamics of the process is observable in the visual strategies is not understood at the moment. In the following studies we attempted to answer these questions by considering the temporal properties of eye-tracking data.

4.4 P4: Levels of analysis of sparse eye-tracking data

4.4.1 Background and aims

Because the previous experiments did not provide sufficient evidence for the presumed differences in the visual attention patterns during comprehension, we further considered the challenges of eye-tracking data analysis. Paper (**P4**) deals with the efficiency of conventional analysis methods as applied in eye-tracking research: there is a trade-off between increasing the number of participants to ensure the statistical power of the tests and the feasibility of doing so.

The previous studies have shown that eye-tracking data in programming can easily become sparse. In addition, the static measures of the use of different areas of interest are not sufficient to evaluate the visual strategies involved in programming. In (**P4**) we therefore propose and discuss two solutions to these two problems.

4.4.2 Results

A review of eye-tracking studies in programming that included some of the studies presented in this thesis (**P2**, **P3**) has shown the current research practices in studying the role of visual attention during programming. With few exceptions, the previous attempts to evaluate the visual attention strategies of programmers tended to employ a factorial design and compared the resulting eye-tracking measures using a parametric statistical procedure. Only a few studies have taken a visual approach to the analysis of the visual attention, by, for example, analyzing the visual attention patterns of a single programmer.

The review shows that a typical eye-tracking study in programming involves about ten to sixteen participants (thirteen on the average). In most of the studies contained in the sample a relatively short program was used and the participants were given a task to comprehend or to debug the program. Similar to usability eye-tracking studies, researchers divide the whole interface into areas of interest. The eye-tracking measures are recorded and aggregated with respect to these areas and the parametric statistical procedures are conducted to compare the measures between the areas. The average duration of a session from which these measures are constructed is about 13.5 minutes.

In (**P4**) we propose that studies of visual attention in programming should take a visual approach to analysis and that the eye-tracking data-sets can be reframed so that a binomial test can be conducted to investigate trends in eye-tracking data.

The eye-tracking data, we suggest, can be visually analyzed on at least three levels of detail. On the lowest level, raw eye-position data can be plotted against time and stimulus. In the domain of eye-tracking studies of programming, this approach has rarely been used, but has recently been introduced by Uwano et al. (2006). We make use of this approach to demonstrate that the same long-term eye-tracking measure can represent two markedly different strategies during programming and vice-versa. However, when plotted over time, the locations of the point of gaze show that the differences in how two programmers allocate their attention happen on a finer scale.

A conventional plot of aggregated eye-tracking measures has been used in a variety of studies. However, in this paper we propose to arrange the eye-tracking measures in a series of trials and plot them in time for each of the participants. This arrangement allows for a single-case type of analysis of visual strategies.

On a hypothetical, but authentic, data-set we demonstrated that it is hard to satisfy the assumptions of parametric statistical procedures when eye-tracking data become sparse. Such analysis of sparse data can result in either low statistical power or inadequate evaluation of the visual strategies. We, therefore, propose to reframe the original data set to construct a new perspective on the eye-tracking data. We then also suggest that a binomial test might be one of the alternatives to the conventional procedures to evaluate visual strategies and we discuss the advantages and disadvantages of this method.

4.4.3 Discussion

Perhaps due to the roots in research methods of psychology, modern eye-tracking studies of programming tend to employ hypothesis-testing designs. These research settings inherently lend themselves to quantitative data and to parametric statistical analyses; mostly used are repeated-measures procedures that compare the means and variances of two or more groups of measures.

Basili et al. (1999) state that experimentation in software engineering is difficult and that carrying out empirical work is complex and time consuming. This seems to be true for conducting and analysing experiments that employ eye-tracking. Automation of data extraction and analysis, together with the reductionist approach to the situation, seem, at first, to be the efficient ways of dealing with the complexity and increased labor.

By adopting the hypothesis testing designs and procedures, eye-tracking researchers, however, "adhere" to the traditions of that framework. This, on one hand, brings about the advantage being able to process large volumes of data automatically, but it also presents 1) danger of providing a too-coarse view on the strategies, 2) threads of underpowered statistical results, and 3) problems when data sets become sparse. In (**P4**) we discussed the last two challenges and proposed some alternative ways to analyze the eye-tracking data. While not necessarily novel, these approaches provide ways of triangulating the data and dealing with the challenges of sparseness.

The first problem with the conventional analysis approach, the too-coarse level of detail in strategies in programming, has been tackled first in (**P3**) by deconstructing part of the interface into smaller areas. The approach suggested in the current report requires deconstructing the data set into shorter segments in time and computing new eye-tracking measures of the original metrics from these shorter segments.

Although the finer level of analysis of visual attention strategies in programming brings new insights, its main disadvantage – as it has been argued also in the previous discussion due to the inefficient software tools – lies in the excessive amount of efforts researcher has to put into processing the data. Also, how to segment the data sets and what should be the boundaries of the segmentation are tasks not well understood at the moment; the following investigations are meant to answer this question.

The unequal number of trials when participants voluntarily engage in performing some activity results in missing data and, hence, results in sparse data sets. By reframing the original eye-tracking data into series of trials, we presented a method for coping with these two intertwined problems inherent in the analysis of data from programming studies.

The measures of proportional or total fixation time or the number of fixations are often regarded as indicators of visual attention to spend time on a certain area and of the importance of an area. An implicit proposal of (**P4**) is that the ratio of this measure for two areas can therefore reflect *the relative importance of one area over another.* In addition, when these ratios are sampled during a session and arranged into a series, the resulting series of proportions can be analyzed for trends. The resulting measure, however, should not be confused with the proportion of time on each area of interest, and to our knowledge, it has not been previously presented in the related literature.

4.5 P5: Role of representations as reflected in eye-tracking measures during program animation

4.5.1 Background and aims

In paper (**P5**), we take a different approach to the analysis of eye-tracking data and also attempt to deal with the sparse-properties of eye-tracking data. Adopting the proposed analysis approach from (**P4**), we reframed the data from (**P3**) and conducted a new analysis. We thus perform triangulation in the sense mentioned in Sharp et al. (2007, p. 293): "*Triangulation is a strategy that entails ... using more than one data analysis approach on the same set of data*".

A motivation of this study was to find out what are the roles of representations during different stages of comprehension and whether and how these roles change and develop. The importance of a representation was operationalized as the proportional time spent visually attending to the representation. To answer the question whether about the relative importance of the two main representations (i.e. the code and the visualization) changes over time, we employed the binomial test to investigate prevailing trends in the series of ratios of fixation times. In addition, the trends in the frequency of switches of visual attention and in the mean fixation durations were analyzed. The two other areas of interest, the area containing the control buttons and the output of the program, were not included in this analysis.

4.5.2 Results

The whole population of participants was divided into two groups, depending on the number of times the visualization of the program execution was replayed. This way of grouping resulted in significant differences in expertise between the groups. Because of the findings from (**P3**) that suggested that there is an effect of the target program on visual attention patterns, the visual attention data from the two programs were analyzed separately.

Only the data from the less experienced group were transformed into series of trials; however, the data from the remaining more-experienced participants were analyzed for possible differences between the target programs. These findings are summarized first.

More experienced participants split their attention between the code and visualization in approximately the same way regardless of the target program. The ratio of the total fixation time spent on the code to the total fixation time spent on the visualization was slightly less than one during animations of both of the programs, but did not significantly differ between the two programs.

Considering the dynamics of visual attention switching between any of the areas of interest, more experienced participants exhibited about same attention switching behavior during both of the target programs.

The overall mean fixation duration was shown not to be significantly affected by the program being comprehended and the measure on each of the areas of interest was about same between the programs. However, ANOVA analysis has shown an effect of area of interest. The area containing the evaluations of expressions was attended with the longest mean fixation duration, while the constants were attended using the shortest fixation duration. The code area was attended with the fixations whose mean durations were under the overall mean fixation durations.

The data sets of the less experienced group were transformed into 28 and 37 trials, respectively, for each of the two target program animations. The ratio of the total fixation time on code to the total fixation time on the animation has shown an increasing trend in both of the programs, indicating that the less experienced group spent more time on the code than on the animation during the later phases of the comprehension task.

The frequency with which the less experienced participants switched their attention between the areas, on the other hand, shown that there is a decreasing trend in both of the programs.

Finally, the analysis of the mean fixation durations focused on possible trends in the measure for each of the areas of interest and for the whole display. The only statistically important decreasing trends were found in the mean fixation durations on the areas containing the visualizations of methods and instances in one of the programs. Other trends were not statistically significant; however, it can be observed that the mean fixation duration increased on the source code of the program.

4.5.3 Discussion of findings

When working with multiple representations, one of the questions that need to be answered is whether and how the role of representations changes as programmers try to build the mental model of a program. Compared to the previous analyses of visual attention in programming that described the overall distribution of fixations and their transitions between the areas for the whole process, in this paper we analyzed the trends in the eye-tracking measures during the process. Thus, in addition to the previous static views on representation use during programming, this study introduced an alternative perspective on the dynamics of visual attention strategies.

Different stages of the comprehension process were delimited by subsequent replays of the visualization. The resulting segments were analyzed using the approach suggested in (**P4**). The analysis of these series suggested that the use of different representations of the program develops during subsequent uses of visualization. For example, at the beginning of the comprehension, less-experienced participants used the graphical representation more than during the later stages of comprehension. In addition, the dynamics visual attention switching of the less experienced participants can be characterized as decreasing. That means, at the beginning of comprehension these programmers were performing higher number of switches per minute.

Taken together, the findings could reflect a difference in how and when the multiple representations are used and integrated. More-experienced programmers integrated new information about the program at the later stages of comprehension, after they had visually attended the source code. On the other hand, less-experienced programmers first made more use of the graphical representation while trying to coordinate it with the textual representation through higher frequencies of switching. At the end of the task, the less experienced programmers concentrated more on the source code while not switching frequently to other available representations.

4.5.4 Methodological discussion

The findings presented here complement the previous knowledge on the visual strategies during comprehension with multirepresentational programming environments. Yet, they further exemplify the need for an increased granularity of temporal eye-tracking data analysis.

While the analysis method based on the binomial distribution seem to be useful in studies of programming, it also can be applied in other domains in which trends and the dynamics of the visual strategies need to be evaluated.

4.6 P6: Temporal properties of eye-tracking data during debugging with static representations

4.6.1 Background and aims

In (**P6**) we further investigate our claim from (**P5**) that argues that *"the comprehension process ... cannot be effectively examined by studying long-term averages [of eye-tracking measures]"*. In this paper we, therefore, further investigated this issue and examined the temporal changes in eye-tracking measures during debugging with multiple available representations.

Many related studies approached behavioral measures from the reductionist perspective, where the reductionism takes the form of computing the long-term average values at the end of an experiment and supposing that the measures provide information about the processes that generated the data. The findings presented in the previous papers, however, indicate that the temporal granularity of this approach is not sufficient. Therefore, some form of segmentation of the eye-tracking data and a finer-level analysis should be conducted to reveal the visual attention strategies as the programmers work with the environment.

The segmentation strategy proposed in (**P5**) is based on the boundaries between different tasks. In the present paper the segments are determined based on fixed time intervals. A portion of the experimental data from (**P1**) served as a case study and has been further analyzed using this segmentation strategy. Each of the whole ten-minute debugging sessions was manually divided up into five two-minute intervals to bring increased temporal detail into the analysis. Our motivation for doing so was to explore the temporal properties of eye-tracking data.

From the original data set, two groups of participants with less and more experience in programming were formed. The result of this assignment materialized not only as the statistically significant differences in experience levels, but also in performance. A summary of the analysis of the eye-tracking data of the two groups is presented in the following section.

4.6.2 Results

The analysis of proportional fixation times showed a statistically significant effect of segment on the measure, an effect of areas of interest on the measure, and an effect of experience. An interaction effect of segment and experience was not significant; however, there was an interaction between segment and area.

A line plot visualizing the proportional fixation times during the five segments showed that there were differences in how the two groups paid attention to different representations during the session. The textual representation was attended to the most; its role, however, changed during the process, during which the experts were using it from about 75% of the total fixation time to about 97% of the time.

The use of the graphical representation negatively mirrored the use of the textual representation, particularly evident in the regular patterns of novice programmers. While the plots of the use of the textual and the graphical representations seem to be similar for the two groups and appear to be constantly offset most of the time, the main characteristic that separates the two experience groups is the use of the output. Expert programmers gradually increased their attention to the output of the program, reaching up to approximately 10% of the overall fixation time at the last phase of debugging.

The analysis of the switching behavior was conducted for the overall frequency of switches and for frequency of switches with respect to the type of switch. The main effect of segment on the overall switching frequency was statistically significant, however, the effect of experience was not. The line plot of the overall switching frequency showed that the two groups exhibited similar behavior during the first three phases of debugging. Both groups began with a higher frequency of about eight switches per minute and dropped to about two switches per minute in the second segment. During the third phase the frequency of switching returned to its initial amplitude.

During the last two phases, however, expert programmers again increased the frequency of switching behavior up to about nine switches per minute at the end of the session. Novice programmers, at the same time, first decreased the switching to finish the debugging with a slightly increased frequency of switches.

The breakdown of the overall switching behavior into three major types of switches provided an explanation of the increased switching activity of the expert group toward the end of the debugging session. The interaction between the segment of a session, type of switch, and experience turned out to be statistically significant. While both groups frequently switched their attention between textual and graphical representations, expert programmers also began to exhibit switches between the source code and output during the later phases of the task.

4.6.3 Discussion of findings

Segmentation of the eye-tracking data into shorter intervals allowed us to form a clearer picture of how programmers' visual strategies during debugging change. In summary, the three views on the visual attention patterns (i.e. the proportional time spent on each representation, the overall- and the detailed visual attention switching frequencies) provide information about the development of strategical differences in novice and expert coordination activities.

Novice programmers engaged in two visual strategies: either switching frequently between the code and visualization or output or focusing mainly on code and having a low frequency of switches to other representations. Experts, in addition to these two strategies, also exhibited a high-frequency of switching between code and output and increased attention to the output areas at later phases of debugging. This finding suggests that experts, more than novices, try to integrate the available information and attempt to relate the source code of a program to the output of the program.

The findings also suggest that the use of representations as measured by eye-tracking should not be regarded as constant throughout the debugging process. The frequency of visual attention switching ranged from 1 to over nine switches per minute and the proportional fixation time on code ranged from 64% to 97%. Therefore, the temporal analysis of eye-tracking data needs to be conducted to reveal the variety of strategies involved in programming.

4.6.4 Methodological discussion

Segmentation of eye-tracking data during complex problem-solving tasks seems therefore to be not only promising, but also necessary. There are several ways to conduct the segmentation. For example, the segments can be determined by a fixed time interval, as shown in this study. The relative simplicity of this segmentation strategy can be seen as an advantage over other methods based on data, such as the one conducted in (**P5**).

On the other hand, a weakness of the proposed segmentation strategy – with regard to the numerical statistical analysis – is that it seems likely that experts who found more bugs quicker were engaged in other activities at the later phases and, therefore, their visual strategies were different from those of novices. Segments that correspond with task boundaries, such as the points when a bug was found, might, therefore, be better suited for statistical evaluation of user strategies.

While some level of automation of the analysis is supported by the present analysis tools, video replays, manual extraction and hard-coding of the scenes for each new segment delimited by a task-reference point is still necessary. An eye-tracking practitioner then needs to be aware that increasing the number of participants and the temporal and spatial granularity of the analysis might not be feasible.

Finally, we observed that the eye-tracking measure of switching between two areas of interest might not completely capture the richness of programmers' strategies. Particularly during the later phases of debugging, more experienced programmers tended to integrate all three representations by switching not only between two areas of interest, but by traversing through all available representations. The matrix containing the transitions of visual attention between the representations then becomes asymmetrical; we suggest that the asymmetry of the matrix can possibly be a future eye-tracking measure to evaluate user coordination strategies.

4.7 Contribution of the Author

The work presented in this thesis has not been conducted in isolation, but is a result of joint efforts of several contributors. The author of this thesis was the principal author of all publications and was responsible for the final drafting of the papers; however, in some instances it is hard to evaluate the exact division of labor. The author of this thesis was the main writer of the first publication (**P1**), has designed and conducted the experiment and analyzed the resulting data.

In papers (**P2**, **P3**) the author was responsible for designing and conducting the experiment, and analysing the data; Niko Myller's assistance and help throughout the process is acknowledged. The writing of the reporting paper (**P3**) was a joint contribution of several authors. In particular, Niko Myller has contributed to the overall work in many ways. Paper (**P4**) had two authors whose contribution was about equal. Paper (**P5**) is a result of discussion with Justus Randolph, who provided many very useful insights and comments. In paper (**P6**) the author proposed and conducted the analysis of the data and he was the primary writer.

Chapter 5

Summary of the Results

"Insanity: doing the same thing over and over again and expecting different results."
– Albert Einstein

THE contributions of this thesis are many. The two main contributions lie in 1) advancing the methodological aspects of eye-tracking in studies of programming and 2) in extending the body of knowledge about visual strategies during programming with multirepresentational environments.

In particular, supportive evidence was presented for utilizing eye-tracking to study visual attention in programming. We then also argued that approaching the analysis of eye-tracking data sets only by adopting quantitative long-term reductionist approaches might not be sufficient to evaluate the user's visual strategies during complex problem-solving processes. In several research inquiries, we were concerned with gaining a deeper understanding of the problem and to relate visual attention to complex reasoning processes. In those inquiries, we advocated employing a variety of analysis approaches to eye-tracking data and we proposed interpreting the (originally) quantitative eye-tracking data from a more qualitative perspective. Using these methods, we then described the strategies of programmers when comprehending or debugging a program in multirepresentational environments.

5.1 Low-Level and High-Level Eye-Tracking Measures

The findings presented in the previous chapters and in the original publications suggest that the currently employed eye-tracking measures are not sufficient to completely evaluate user's visual strategies during programming. While carrying out a task, programmers engage in a variety of strategies, like reading a source code, coordinating different representations, tracking a bug, searching for a statement, or studying documentation. Elementary measures, such as the time spent attending to the source code, might be useful to provide a static view on these strategies. This thesis do not attempt to dismiss them. However, the richness and dynamics of the strategies over time might not be visible when areas of interest match the most general level of detail of the programming interface and the elementary measures are aggregated using a long-term time-frame. One of the conclusions of this thesis is that the elementary eye-tracking measures need to be further processed to better facilitate the exploration and identification of the user's strategies during programming.

Therefore, in addition to the classification of Goldberg and Kotval, we suggest arranging eye-tracking measures as belonging to either of two categories that we label low-level and high-level.[1] *Low-level eye-tracking measures* are computed directly on the information about single fixations or saccades and no other information is used. For example, mean fixation duration, total fixation duration, or the total number of fixations, are all low-level measures. Scanpath duration is yet another example of a low-level eye-tracking measure that can be computed without other contextual information, because it "*can be calculated independently of the actual screen layout*" (Goldberg and Kotval, 1998, p. 531).

A *high-level eye-tracking measure*, on the other hand, is a higher-level composed construct, amalgamating the low-level measures and other behavioral information, such as time, task order, or other measures derived from eye-tracking data. For example, the transition matrix can be regarded as a high-level eye-tracking measure, as it contains not only the mere fixations; it attempts to capture the sequential behavior of the visual attention in time and with respect to the areas of interest. When the cells in the transition matrix are further manipulated and analyzed, new, even higher level information on visual behavior can be obtained, as suggested in (**P4**, **P6**).

For example, the counts of the transitions between each of the areas could be related to the time a user was carrying on the task – the counts of transitions could be divided by the duration of the task to obtain the frequency of the switching – to describe the *dynamics* of the attention switching behavior between areas. Concerning the transition matrix measure, Goldberg and Kotval (1998) suggest characterizing the transition matrix by a single value: the number of active cells is divided by the total number of cells in the matrix. Depending on the content, a large value of the resulting measure would indicate a lengthy undirected scanpath.

Graphical visualizations of gaze behavior can be considered as yet another high-level measure of the users' visual strategies. Plots of fixations superimposed on the

[1]Other alternatives to label the measures could be direct and indirect, or independent and contextual.

areas of interest in time can provide information about the distribution of the gaze direction with respect to the stimuli within a given time segment. Static as well as dynamic visualizations of eye-movements (video replays) can be utilized for this approach.

All of the most often used eye-tracking measures that Jacob and Karn (2003) listed belong to the low-level class of eye-tracking measures. We argue that to study visual strategies during complex tasks, these measures, if used alone and approached from the reductionist perspective, are inadequate to aid the researcher in evaluating the link between visual attention and underlying processing. Single measures, without the context of the sequence and task they belong to, and without a link to the related stimuli, provide information only about a static, long-term characteristics of the users' strategies. As we have shown, these strategies often change and evolve during programming activities and, therefore, low-level long-term eye-tracking measures cannot completely evaluate them.

5.2 The Choice of Method

The choice of a particular measurement method and analysis approach might depend on several factors, such as the type of research situation, on the driving research question and analysis concerns, on the size of the sample population, on the length of a unit of analysis, or on other properties of the data-set.

To answer the research questions related to the tools, this research has shown that remote eye-tracking is a superior measurement method to an alternative method to track visual attention in programming. Based on those findings, we recommend the researchers in complex problem-solving domains, such as the psychology of programming, consider employing eye-tracking to study aspects of visual attention.

Conventional quantitative approaches the to automation of the analysis of the visual attention data operate on long-term eye-tracking measures and provide information about the overall distribution of visual attention on interface elements. They, however, are not suitable for guiding a researcher investigating strategic differences in visual attention during complex interaction tasks. In addition, it seems that traditional measures of visual attention, such as fixations and saccades, are not exhaustive enough and are not context-specific enough to guide the eye-tracking researcher wanting to make sense out of eye-tracking data in programming.

To answer the research questions related to the analysis approaches, we therefore suggest two ways of dealing with the problems. The first solution is to design and employ the higher-level eye-tracking measures described above that can facilitate answering the intuitive questions about visual attention patterns. In this thesis, we suggested that eye-tracking measures should be arranged into series and seen as consecutive trials; these series were analyzed for containing trends. This approach helps to overcome the problems of sparse eye-tracking data sets that are common in the type of studies conducted in this thesis.

The second solution suggested by this work is that more detailed and qualitative approaches to analysis have to be employed and combined with the quantitative views that eye-tracking measures naturally support. In particular, segmentation of the data sets has been shown as a way to discover finer level differences in visual strategies. Segmentation of lengthy eye-tracking data sequences and a segmentation of the display into smaller, more semantically distinct areas of interest are just two ways of bringing more detail into the conventional coarse-level analysis.

Combining different methods in HCI research is a usual practice (Sharp et al., 2007) that leads to triangulation of results to achieve a better understanding of a phenomenon. This thesis suggested that combining different analysis methods of eye-tracking data and multiple measures allows for constructing a better picture of the complex processes involved in programming.

To summarize the methodological viewpoint of this work, the studies reported in this thesis show how different eye-tracking techniques, measures and the methods to analyze the data can contribute to the understanding of the link between visual attention and underlying processes in programming. To maximize the contribution of each method, we advise the researchers of visual attention to (with references to the papers that primarily discussed each item):

- Employ the Focus Window Technique with caution, as it tends to interfere with natural strategies (**P1**).
- Approach the analysis of data on multiple levels of detail (**P4**).
- Focus the questions related to the visual attention patterns to also investigate the temporal aspects of the strategies and reframe the data sets into consecutive trials in order to identify the potential trends in use of areas of interest (**P4**, **P5**).
- Employ the ratio of two low-level eye-tracking measures of total fixation time to reveal the relative importance of two areas of interest (**P4**, **P5**).
- Segment the sessions into meaningful shorter fragments (**P6**). In the case of sparse datasets that can appear due to the segmentation, employ the binomial analysis (**P4**).
- Capture and triangulate other contextual information about the interaction and underlying processing to create a more complete picture (**P3**).

5.3 Visual Strategies of Programmers

By applying various data analysis methods, we have gained new insights into the visual strategies of programmers. The new knowledge together with the analysis methods and measures presented here provide the answers to the questions related to visual attention strategies during programming. Therefore, in addition to advancing methodological aspects of eye-tracking, this work contributes to the understanding of the role of visual attention during programming with multiple representations. More specifically, we found out that:

- Mean fixation duration is sensitive to the level of programming experience (**P1**, **P3**).
- Different representations of a program are attended with different mean fixation durations. A source code of a program is typically attended with the longest fixations, while a graphical representation elicits shorter mean fixation durations (**P1**, **P3**).
- Long-term proportional fixation times on different representations during visualization are not affected by experience in programming (**P1**, **P3**).
- The increased number of switches between the representations seem to indicate that that expert programmers integrate more information than less experienced programmers (**P1**, **P6**).
- Short-term proportional fixation times and the frequency of visual attention switching reveal how and when programmers attempt to integrate information from multiple sources (**P5**, **P6**). A detailed analysis of overall visual attention patterns shows that expert programmers tend to first focus on understanding the code (**P5**) and integrate more information mostly at the later phases of comprehension and debugging. Experts, also, tend to attend to the output and relate it to the source code of the program (**P6**).
- More experienced programmers attend more to source code than to other available representations (**P1**, **P3**, **P6**) and also attend to the source code more than novice programmers do (**P3**, **P6**).
- During program visualization, there is an effect of the visualized program on the general visual attention patterns (**P3**). However, more experienced programmers employ similar strategies regardless of a program being comprehended (**P5**).

- Indicated by both long-term quantitative and by qualitative analyses, the visual strategies of experienced programmers are more diverse than the visual strategies of less experienced programmers.

5.4 Recommendations to Eye-tracking Developers

A part of the research conducted in this thesis has been carried out using instruments and tools supplied by the manufacturers of eye-tracking systems. Another part of the research, however, could have not been performed without developing new tools and methods to analyze eye-tracking data from the new perspectives introduced above.

Based on the experiences with conducting eye-tracking research in the domain of dynamic computer displays and based on the findings of this thesis, there are recommendations we believe that manufactures of future eye-tracking systems and tools need to make use of. The most important feature that is currently missing and that would further improve the efficiency of analysis of eye-tracking data is a systematic way for allowing mapping of eye-tracking data to dynamic scenes. The following list also contains other features that the future systems should implement and support, to further facilitate the evaluations of user's visual strategies with dynamic computer displays:

- Allow for dynamic areas of interest that can be created, disabled, moved, and their geometry modified; all operations should be made available during both the recording and the analysis process.

- Provide open ways of mapping dynamic content to the related eye-tracking data. This can be achieved, for example, by allowing user applications to send standardized events to the eye-tracking application to manipulate the areas of interest during the recording time.

- Allow automatic analysis and visualization of the eye-tracking data with respect to the aforementioned events.

- Provide means of building and applying customized analysis and visualization methods and custom contextual eye-tracking measures.

Chapter 6

Conclusion

"You need the willingness to fail all the time. You have to generate many ideas and then you have to work very hard only to discover that they don't work. And you keep doing that over and over until you find one that does work."
– John W. Backus (3.12.1934 - 17.3.2007)

APPLYING a new data collection method requires an understanding of the advantages, possibilities, limitations and challenges of that method. McGrath (1995) sees research methods as "*bounded opportunities*"; that is, a method not only offers opportunities to gather knowledge not available through other methods, but it also has inherent limitations in its use. In this thesis, the methodological issues of eye-tracking as applied to the study of the visual strategies of programmers were discussed.

The research reported in this thesis has presented eye-tracking as a superior technology to track visual attention during programming with multiple representations. A comparison of eye-tracking with the Focus Window Technique has shown the problems associated with restricting the displays by blurring and requiring users to manually indicate the focus of attention.

Although eye-tracking is a tool that does not interfere with the natural strategies of users, it produces essentially the same complex stream of behavioral data as other methods. Even a short recording results in eye-tracking data of large volumes. For example, an average number of fixations in the first study (**P1**) was approximately 1000 fixations per one participant during only the unrestricted display treatment. Processing such mountainous streams of data manually is obviously not feasible. Therefore, there is a need to expand the analysis methods and tools to these situations. One of the ways of dealing with the challenges is to develop automatic approaches to analyze the eye-tracking data.

In this thesis, different approaches to analyzing eye-tracking data and evaluating the visual strategies during programming were discussed. We began our investigation by applying the quantitative approaches to data analysis, as is the tradition in eye-tracking usability studies and in experimental studies of psychology of programming. We then expanded the variety of available analysis methods and argued 1) for employment of a method to evaluate the trends in eye-tracking data sets that can be sparse, 2) for the exploration of temporal properties of visual attention patterns, and 3) for the employment of more qualitative methods to analysis.

Goldberg and Kotval (1999, p. 644), in the context of evaluating users' strategies during search tasks, argue that "*strategy differences are most evident during lengthy (e.g. 10-15 s) tasks*". Using their method, we have investigated the strategical differences in tasks that are considerably longer and more complex. We proposed to explore the multiple levels of detail present in the eye-tracking data that cannot, we argue, be well explored by the purely quantitative approaches. Although not necessary novel, the proposed approaches are not currently present in the repertoire of eye-tracking studies nor are they supported by commonly available analysis tools. Yet, they provide a more detailed level for interpretation and, thereby, allow us to get a deeper understanding of the strategies involved in computer programming.

The analysis methods proposed in this thesis help not only to deal with the challenges that an eye-tracking study of complex problem solving might face, the improvements also help in the other part of the eye-tracking challenge – that is, in interpretation of eye-tracking data. The method proposed in (**P4**) and applied in (**P5**) requires reframing the research questions and data sets so that longer intervals are segmented into shorter units. Similarly, segmentation of eye-tracking data into shorter intervals based on a time frame, as introduced in (**P6**), suggests that the eye-tracking researchers needs to find an appropriate segmentation strategy. This instructive characteristic of the presented analysis methods guarantees that the analysis of eye-tracking data is no longer approached only from the long-term perspective.

When applied to the study of the visual attention strategies during programming with multirepresentational environments, the proposed approaches helped not only in the understanding of what the effects of experience on the strategies are in overall, but also when they materialize. Expert programmers were shown to be better able to integrate more information from the available representations. In particular, they concentrated more on the textual representation at the earlier phases of program understanding. During later phases, they focused on relating the code to the output. Novice programmers, on the other hand, engaged in a limited set of strategies and they seemed to alternate between them. In particular, they either performed a high frequency of visual attention switching between code and graphical representation or they tended to mainly maintain the focus of visual attention on one of them.

6.1 Questions for Future Research

This thesis hints at several directions that future research might take. The notion of higher level eye-tracking metrics is one of them. In (**P6**) we proposed another way of constructing higher-level measures. In particular, we suggested that the degree of asymmetry of a transition matrix of switching frequency between different areas of interest might be the next high-level eye-tracking metric. Although rooted in and specific to the application domain, it is easy to imagine that this metric could also be applied in other fields, such as usability.

The development of new high-level eye-tracking measures, however, requires a further increase in the detail of the analysis by means of temporal and spatial segmentation of the data sets. It is easy to envision the amount of manual labor and time associated with each increase of detail. This progress needs to be accompanied by improved analysis tools; thus, future eye-tracking research needs to carefully consider software tool development.

The problem of identifying the meaningful segmentation strategies is left for future research. An approach that we see promising is allowing for adaptive duration of the data segments, based on events during the process, to account for the variability in the strategies.

In addition to the analysis methods discussed in this thesis, there are also other ways to evaluate systematic differences in visual strategies. For example, Hyönä et al. (2002) and Hyönä and Nurminen (2006) employed a clustering algorithm to find individual differences in reading patterns. It is possible that clustering of data sets produced by programmers during comprehension or debugging might yield new insights into their visual strategies. We suggest that this direction is investigated in future studies.

Our studies indicate that there is an entirely new level of detail in visual attention data during programming that has not been explored before. The studies of programming are, however, only one particular domain we selected to conduct this investigation. Our results have the potential of informing other domains, but future work is needed to extend the findings.

References

Aaltonen, A., Hyrskykari, A., and Räihä, K.-J. (1998). 101 spots, or how do users read menus? In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 132–139, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Basili, V. R., Shull, F., and Lanubile, F. (1999). Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25(4):456–473.

Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M. (2005). Effects of experience on gaze behaviour during program animation. In *Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05)*, pages 49–61, Brighton, UK.

Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M. (2006). Analyzing Individual Differences in Program Comprehension. *Technology, Instruction, Cognition and Learning*, 3(3-4):205–232.

Bednarik, R. and Randolph, J. (2008). Studying cognitive processes in program comprehension: Levels of analysis of sparse eye-tracking data. In Hammoud, R., editor, *Passive Eye Monitoring: for Safety, Security, Communications, Medical and Web Applications*. Springer.

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

Bednarik, R. and Tukiainen, M. (2007a). Analysing and interpreting quantitative eye-tracking data in studies of programming: Phases of debugging with multiple representations. In *Proceedings of the 19th Annual Psychology of Programming Interest Group Workshop (PPIG'07)*, pages 158–172, Joensuu, Finland.

Bednarik, R. and Tukiainen, M. (2007b). Validating the restricted focus viewer: A study using eye-movement tracking. *Behavior Research Methods*, 39(2):274–282.

Blackwell, A. F. (2002). First steps in programming: A rationale for attention investment models. In *HCC '02: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)*, page 2, Washington, DC, USA. IEEE Computer Society.

Blackwell, A. F., Whitley, K. N., Good, J., and Petre, M. (2001). Cognitive factors in programming with diagrams. *Artificial Intelligence Review*, 15(1/2):95–114.

Bower, G. H. and Clapper, J. P. (1989). Experimental methods in cognitive science. pages 245–304.

Branch, J. L. (2000). Investigating the information-seeking processes of adolescents: The value of using think alouds and think afters. *Library and Information Science Research*, 22(4):371–392.

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. *Int. Journal of Man-Machine Studies*, 9:737–751.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *Int. Journal of Man-Machine Studies*, 18:543–554.

Burkhardt, J., Détienne, F., and Wiedenbeck, S. (2002). Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering*, 7(2):115–156.

Byckling, P., Kuittinen, M., Nevalainen, S., and Sajaniemi, J. (2004). An inter-rater reliability analysis of good's program summary analysis scheme. In *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004)*, pages 170–184, Institute of Technology Carlow, Ireland.

Carpenter, R. H. S. (1988). *Movements of the Eyes. (2nd ed.).* Pion, London, UK.

Carroll, J. M. (2003). *HCI Models, Theories, and Frameworks.* Morgan Kaufman Publishers, San Francisco.

Chi, M. T. H. (1997). Quantifying qualitative analyses of verbal data: a practical guide. *Journal of Learning Sciences*, 6(3):271–315.

Corritore, C. L. and Wiendenbeck, S. (2001). An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Human-Computer Studies*, 54(1):1–23.

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

Creswell, J. W. (2003). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* SAGE Publications. second edition.

Crosby, M. and Peterson, W. (1991). Using eye movements to classify search strategies. In *Proceedings of the Human Factors Society 35th Annual Meeting*, pages 1476–1480.

Crosby, M. E. and Stelovsky, J. (1990). How do we read algorithms? A case study. *IEEE Computer*, 23(1):24–35.

Détienne, F. (2002). *Software Design - Cognitive Aspects.* Springer-Verlag, Inc., London, UK.

Duchowski, A. T. (2003). *Eye Tracking Methodology: Theory & Practice.* Springer-

Verlag, Inc., London, UK.

Duchowski, A. T., Medlin, E., Cournia, N., Gramopadhye, A., Melloy, B., and Nair, S. (2002). 3D eye movement analysis for VR visual inspection training. In *ETRA '02: Proceedings of the symposium on Eye tracking research & applications*, pages 103–110, New York, NY, USA. ACM Press.

Duchowski, A. T., Shivashankaraiah, V., Rawls, T., Gramopadhye, A. K., Melloy, B. J., and Kanki, B. (2000). Binocular eye tracking in virtual reality for inspection training. In *ETRA '00: Proceedings of the symposium on Eye tracking research & applications*, pages 89–96, New York, NY, USA. ACM Press.

Ericsson, K. A. and Simon, H. A. (1984). *Protocol analysis: Verbal reports as data.* Braford Books/MIT Press, Cambridge, MA.

Fitts, P. M., Jones, R. E., and Milton, J. L. (1950). Eye movement of aircraft pilots during instrument-landing approaches. *Aeronautical Engineering Review*, (9):24–29.

Fix, V., Wiedenbeck, S., and Scholtz, J. (1993). Mental representations of programs by novices and experts. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 74–79, New York, NY, USA. ACM Press.

Futrelle, R. P. and Rumshisky, A. (2001). Discourse structure of text-graphics documents. In *Proceedings of 1st International Symposium on Smart Graphics*, New York, NY, USA. ACM Press.

Gilmore, D. J. (1990). Methodological issues in the study of programming. In Hoc, J. M., Green, T. R. G., Samurcay, R., and Gilmore, D. J., editors, *The Psychology of Programming*, pages 83–98. Academic Press.

Goldberg, J. and Kotval, X. P. (1998). Eye movement-based evaluation of the computer interface. In Kumar, S. K., editor, *Advances in Occupational Ergonomics and Safety*, pages 529–532. IOS Press.

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

Goldberg, J. H. and Wichansky, A. M. (2003). Eye tracking in usability evaluation: A practitioner's guide. In Hyönä, J., Radach, R., and Deubel, H., editors, *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, pages 493–516. Elsevier Science.

Good, J. and Brna, P. (2004). Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. *International Journal of Human Computer Studies*, 61(2):169–185.

Gugerty, L. and Olson, G. (1986a). Debugging by skilled and novice programmers. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 171–174, New York, NY, USA. ACM Press.

Gugerty, L. and Olson, G. M. (1986b). Comprehension differences in debugging by skilled and novice programmers. In *First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, pages 13–27.

Hoc, J. M., Green, T. R. G., Samurcay, R., and Gilmore, D. J. (1990). *The Psychology of Programming.* Academic Press.

Hyönä, J., Lorch, R., and Kaakinen, J. K. (2002). Individual differences in reading to summarize expository text: Evidence from eye fixation patterns. *Journal of Educational Psychology*, 94:44–55.

Hyönä, J. and Nurminen, A.-M. (2006). Do adult readers know how they read? evidence from eye movement patterns and verbal reports. *British Journal of Psychology*, 97:31–50(20).

Jacob, R. J. K. (1991). The use of eye movements in human-computer interaction techniques: what you look at is what you get. *ACM Transactions of Information Systems*, 9(2):152–169.

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 Hyönä, J., Radach, R., and Deubel, H., editors, *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, pages pp. 573–605. Elsevier Science.

Jansen, A. R., Blackwell, A. F., and Marriott, K. (2003). A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, and Computers*, 35(1):57–69.

John, B. E. (2003). Information processing and skilled behavior. In Carroll, J. M., editor, *HCI Models, Theories, and Frameworks*, pages 55–101, San Francisco. Morgan Kaufman Publishers.

Josephson, S. and Holmes, M. E. (2002). Visual attention to repeated internet images: testing the scanpath theory on the world wide web. In *ETRA '02: Proceedings of the 2002 symposium on Eye tracking research & applications*, pages 43–49, New York, NY, USA. ACM Press.

Just, M. A. and Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology*, 8:441–480.

Just, M. A. and Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review*, 87(4):329–354.

Ko, A. J. and Myers, B. A. (2004). Designing the whyline: a debugging interface for asking questions about program behavior. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 151–158, New York, NY, USA. ACM Press.

Koenemann, J. and Robertson, S. P. (1991). Expert problem solving strategies for program comprehension. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 125–130, New York, NY, USA. ACM Press.

Law, B., Atkins, M. S., Kirkpatrick, A. E., and Lomax, A. J. (2004). Eye gaze patterns differentiate novice and experts in a virtual laparoscopic surgery training environment. In *ETRA'2004: Proceedings of the Eye tracking research & applications symposium*, pages 41–48, New York, NY, USA. ACM Press.

Layzell, P. J., Champion, R., and Freeman, M. J. (1993). Docket: program comprehension-in-the-large. In *Proceedings of IEEE Second Workshop on Program Comprehension*, pages 140–148.

Lethbridge, T. C., Sim, S. E., and Singer, J. (2005). Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, 10(3):311–341.

Letovsky, S. (1986). Cognitive processes in program comprehension. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 58–79, Norwood, NJ, USA. Ablex Publishing Corp.

Littman, D. C., Pinto, J., Letovsky, S., and Soloway, E. (1986). Mental models and software maintenance. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 80–98, Norwood, NJ, USA. Ablex Publishing Corp.

Lukka, K. (2003). The constructive research approach. In Ojala, L. and Hilmola, O.-P., editors, *Case study research in logistics. Publications of the Turku School of Economics and Business Administration, Series B 1*, pages 83–101.

MacKenzie, I. S. (2003). Motor behavior models for human-computer interaction. In Carroll, J. M., editor, *HCI Models, Theories, and Frameworks*, pages 27–54, San Francisco. Morgan Kaufman Publishers.

Majaranta, P. and Räihä, K.-J. (2002). Twenty years of eye typing: systems and design issues. In *ETRA '02: Proceedings of the symposium on Eye tracking research & applications*, pages 15–22, New York, NY, USA. ACM Press.

Matin, E. (1974). Saccadic suppression: a review and an analysis. *Psychological Bulletin*, 81(12):889–917.

McGrath, J. E. (1995). Methodology matters: doing research in the behavioral and social sciences. In *Human-computer interaction: toward the year 2000*, pages 152–169, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Miller, J. (2004). Statistical significance testing – a panacea for software technology experiments? *Journal of Systems and Software*, 73(2):183–192.

Miller, S. (1984). *Experimental Design and Statistics.* Routledge, London, UK. Second edition.

Monk, A., Nardi, B., Gilbert, N., Mantei, M., and McCarthy, J. (1993). Mixing oil and water?: Ethnography versus experimental psychology in the study of computer-mediated communication. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 3–6, New York, NY, USA. ACM Press.

Murphy, G. C., Walker, R. J., and Baniassad, E. L. A. (1999). Evaluating emerging software development technologies: Lessons learned from assessing aspect-oriented programming. *IEEE Trans. Software Eng.*, 25(4):438–455.

Nevalainen, S. and Sajaniemi, J. (2004). Comparison of three eye tracking devices in psychology of programming research. In *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004)*, pages 151–158, Institute of Technology Carlow, Ireland.

Nevalainen, S. and Sajaniemi, J. (2005). 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)*, pages 77–91, Brighton, UK.

Nevalainen, S. and Sajaniemi, J. (2006). An experiment on short-term effects of animated versus static visualization of operations on program perception. In *ICER '06: Proceedings of the 2006 international workshop on Computing education research*, pages 7–16, New York, NY, USA. ACM Press.

Nielsen, J., Clemmensen, T., and Yssing, C. (2002). Getting access to what goes on in people's heads? Reflections on the think-aloud technique. In *Proceedings of The Second Nordic Conference on Human-Computer Interaction (NordiCHI'02)*, pages 101–110, New York, NY, USA. ACM Press.

O'Brian, M. P., Buckley, J., and Shaft, T. M. (2004). Expectation-based, inference-based, and bottom-up software comprehension. *Journal of Software Maintenance and Evolution: Research and Practice*, 16:427–447.

Ormerod, T. C., Mariani, J. A., Morley, N. J., Rodden, T., Crabtree, A., Mathrick, J., Hitch, G., and Lewis, K. (2004). Mixing research methods in HCI: Ethnography meets experimentation in image browser design. In *EHCI/DS-VIS*, pages 112–128.

Owen, S., Brereton, P., and Budgen, D. (2006). Protocol analysis: a neglected practice. *Communications of the ACM*, 49(2):117–122.

Pan, B., Hembrooke, H. A., Gay, G. K., Granka, L. A., Feusner, M. K., and Newman, J. K. (2004). The determinants of web page viewing behavior: an eye-tracking study. In *ETRA '04: Proceedings of the 2004 symposium on Eye tracking research & applications*, pages 147–154, New York, NY, USA. ACM Press.

Pennington, N. (1987). Comprehension strategies in programming. In *Empirical studies of programmers: second workshop*, pages 100–113, Norwood, NJ, USA. Ablex Publishing Corp.

Rajlich, V. (1994). Program reading and comprehension. In *Proceedings of Summer School on Engineering of Existing Software*, pages 161–178, Dipartimento di Informatica, University of Bari, Italy.

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

Renshaw, J. A., Finlay, J., and Webb, N. (2006). Getting a measure of satisfaction from eyetracking in practice. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1723–1726, New York, NY, USA. ACM Press.

Renshaw, J. A., Finlay, J. E., Ward, R. D., and Tyfa, D. (2004). Understanding visual influence in graph design through temporal and spatial eye movement characteristics. *Interacting with Computers*, 16:557–578.

Romero, P., Cox, R., du Boulay, B., and Lutz, R. (2002a). Visual attention and representation switching during java program debugging: A study using the restricted focus viewer. In *DIAGRAMS '02: Proceedings of the Second International Conference on Diagrammatic Representation and Inference*, pages 221–235, London, UK. Springer-Verlag.

Romero, P., du Boulay, B., Cox, R., and Lutz, R. (2003a). Java debugging strategies in multi-representational environments. In *15th Annual Workshop of the Psychology of Programming Interest Group (PPIG'03)*, pages 412–434.

Romero, P., du Boulay, B., Lutz, R., and Cox, R. (2003b). The effects of graphical and textual visualisations in multi-representational debugging environments. In *Proceedings of 2003 IEEE Symposia on Human Centric Computing Languages and Environments*, pages 236–238, Washington, DC, USA. IEEE Computer Society.

Romero, P., Lutz, R., Cox, R., and du Boulay, B. (2002b). 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)*, pages 207–214, Washington, DC, USA. IEEE Computer Society.

Räsänen, M. and Nyce, J. M. (2006). A new role for anthropology?: rewriting "context" and "analysis" in HCI research. In *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction*, pages 175–184, New York, NY, USA. ACM Press.

Salvucci, D. D. and Anderson, J. R. (2001). Automated Eye-Movement Protocol Analysis. *Human-Computer Interaction*, 16:39–86.

Sharp, H., Preece, J., and Rogers, Y. (2007). *Interaction Design: Beyond human-computer interaction. Second edition.* John Wiley & Sons, Inc., New York, NY, USA.

Sheil, B. A. (1981). The psychological study of programming. *ACM Comput. Surv.*, 13(1):101–120.

Shneiderman, B. (1986). Empirical studies of programmers: the territory, paths, and destination. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 1–12, Norwood, NJ, USA. Ablex Publishing Corp.

Sibert, L. E. and Jacob, R. J. K. (2000). Evaluation of eye gaze interaction. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–288, New York, NY, USA. ACM Press.

Stein, R. and Brennan, S. E. (2004). Another person's eye gaze as a cue in solving programming problems. In *ICMI '04: Proceedings of the 6th Int. Conference on Multimodal Interfaces*, pages 9–15, New York, NY, USA. ACM Press.

Tarasewich, P. and Fillion, S. (2004). Discount eye tracking: The enhanced restricted focus viewer. In *Proceedings of 2004 Americas Conference on Information Sys-*

tems, pages 1–9, New York, NY, USA.

Tarasewich, P., Pomplun, M., Fillion, S., and Broberg, D. (2005). The enhanced restricted focus viewer. *International Journal of Human-Computer Interaction*, 19(1):35–54.

Underwood, G., Chapman, P., Brocklehurst, N., Underwood, J., and Crundall, D. (2003). Visual attention while driving: Sequences of eye fixations made by experienced and novice drivers. *Ergonomics*, 46(6):629–646.

Uwano, H., Nakamura, M., Monden, A., and ichi Matsumoto, K. (2006). Analyzing individual performance of source code review using reviewers' eye movement. In *ETRA '06: Proceedings of the 2006 symposium on Eye tracking research & applications*, pages 133–140, New York, NY, USA. ACM Press.

van den Haak, M., Jong, M. D., and Schellens, P. J. (2003). Retrospective vs. concurrent think-aloud protocols: testing the usability of an online library catalogue. *Behaviour and Information Technology*, 22(5):339–351.

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, 23(5):459–494.

von Mayrhauser, A. and Lang, S. (1999). A coding scheme to support systematic analysis of software comprehension. *IEEE Transactions on Software Engineering*, 25(4):526–437.

von Mayrhauser, A. and Vans, A. M. (1995). Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55.

von Mayrhauser, A. and Vans, A. M. (1996). Identification of dynamic comprehension processes during large scale maintenance. *IEEE Transactions on Software Engineering*, 22(6):424–437.

Weinberg, G. and Shulman, E. (1974). Goals and performance in computer programming. *Human Factors*, 16(1):70–77.

Yoon, D. and Narayanan, N. H. (2004). Mental imagery in problem solving: an eye tracking study. In *ETRA '04: Proceedings of the 2004 symposium on Eye tracking research & applications*, pages 77–84, New York, NY, USA. ACM Press.

Original Publications

P1.

1

Bednarik, R., Tukiainen, M.: **Validating the Restricted Focus Viewer: A Study Using Eye-Movement Tracking.** Behavior Research Methods, 39(2), 2007, pp. 274-282.

Behavior Research Methods
2007, 39 (2), 274-282

Validating the Restricted Focus Viewer: A study using eye-movement tracking

Roman Bednarik and Markku Tukiainen
University of Joensuu, Joensuu, Finland

Investigation of cognitive processes and visual attention during problem-solving tasks is an important part of understanding human reasoning. Eyetracking technology has proven to have many benefits in revealing visual attention patterns. However, the high price of accurate eyetrackers and the difficulties associated with using them represent major obstacles to their wider application. Therefore, previous studies have sought to find alternatives to eyetracking. The Restricted Focus Viewer (RFV) brings a small part of an otherwise blurred display to the focus of visual attention: A user controls what part of the screen is in focus by using a computer mouse and explicitly selecting the area to be shown in focus. Recently, some studies have employed the RFV to investigate cognitive behavior of users, and some researchers have even enhanced the tool to study usability. We replicated a previous RFV-based study while also recording gaze data. We compared the attention allocation in time and space as reported by the RFV and an eyetracker. Further, we investigated the effects of RFV's display blurring on the visual attention allocation of 18 novice and expert programmers. Our results indicate that the data obtained from the two tools differ. Also, the RFV-blurring interferes with the strategies utilized by experts, and has an effect on fixation duration. However, task performance was preserved.

Researchers who investigate cognitive processing during reasoning tasks have several options for how to get insights into the behavior and strategies exhibited by the participants. Especially in situations in which the stimulus is visual and the reasoning is related (and dependent on it), eyetracking systems have proven useful in revealing the patterns of visual attention during the task. Typical examples of successful applications of eye-movement tracking include studies relating eye-movement patterns to cognitive processes (Just & Carpenter, 1976), studies on reading (e.g., Rayner, 1998), and studies that investigated differences between novices and experts with regard to eye-movement patterns (e.g., Hyönä, Lorch, & Kaakinen, 2002; Law, Atkins, Kirkpatrick, & Lomax, 2004). However, the relatively high price of accurate eyetracking equipment, and other issues (such as drift, a need for calibration, and a certain level of obtrusiveness), have prevented a wider application. To remedy some of the problems and limitations typical of eyetracking, researchers have sought to develop cheap, yet accurate, alternatives.

Recently, the Restricted Focus Viewer (RFV; Jansen, Blackwell, & Marriott, 2003), a tool that blurs the display and restricts users to only a small focused point, has been introduced as an alternative tool to track visual attention, and it has also been used in various studies. The RFV has been validated in two experiments run by Jansen et al. (2003); however, these validations involved only relatively simple reasoning with visual stimuli. In other studies, the RFV-based technology has been applied (1) to discover strategies of participants debugging computer programs, with the aid of multiple and linked visual representations of the programs (Romero, Cox, du Boulay, & Lutz, 2002; Romero, du Boulay, Cox, & Lutz, 2003; Romero, du Boulay, Lutz, & Cox, 2003; Romero, Lutz, Cox, & du Boulay, 2002), (2) to investigate the issues of usability of hyperlinked documents (Tarasewich & Fillion, 2004), and (3) in a study of shifts of visual attention during integration of text and graphics (Futrelle & Rumshisky, 2001). For the purposes of visual attention tracking during complex reasoning tasks such as computer program debugging, the use of the RFV-based approach has been questioned (Bednarik & Tukiainen, 2004).

In this article, we present a replication of a previous study by Romero and colleagues, in which visual attention of computer programmers was recorded using the RFV. In our study, together with the RFV-based measurement of visual attention, we used a remote eyetracker to collect the patterns of eye movements of participants who were debugging computer programs. Although the RFV has already been validated in previous studies, this validation has involved only relatively simple tasks. It remains an open question whether the behavior of experimental participants during cognitively more demanding tasks is influenced by having to manually move the only focused spot within otherwise blurred visual stimuli.

This article contributes to the body of knowledge in many ways. First, it is important in validating and cross-validating the tools employed in the experiments investi-

R. Bednarik, bednarik@cs.joensuu.fi

gating visual attention and cognitive processing in general. Although the RFV technology is not supposed to replace eye-movement-tracking systems, it aims to provide accurate information about visual attention allocation during reasoning with visual stimuli. By replicating one of the previous studies in which the RFV was employed, we compare the RFV-based measurement with the data collected using a remote eye-movement tracker. Second, we investigate a possible effect of the display-blurring technique on the behavior and performance of experimental participants. Recently, some researchers have started to use the RFV for recording visual attention patterns during tasks involving complex reasoning. We provide important information about (1) how the actual implementation of the RFV technique influences the participants, (2) the data collected, and therefore also (3) the conclusions drawn from the experiments.

Previous Related Studies

The RFV (Jansen et al., 2003) has been developed as an alternative to eye-movement-tracking technologies. One of the main claimed advantages of RFV is that it allows for an automated collection of participants' foci of visual attention. The RFV blurs the stimulus image and displays it on a computer screen, allowing the participant to see only a limited focused region at a time. In order to have another part of the stimuli focused, the participant has to move the focused region using the computer mouse. The RFV then records the moves over the stimuli, which are stored for later analysis. The tool collects time-stamped data about mouse and keyboard events, the currently focused region's index, the total durations of sessions, and other events. Voice protocols can be recorded along with the interaction data. The RFV is of course not capable of collecting visual attention data when the blurring is turned off.

Analysis and interpretation of the data recorded by RFV, and the data's relation to the investigated task, are (as is also true for eyetracking) up to the researcher. Usually, the so-called areas of interest (AOIs) are defined within the interface, and several metrics can be computed—for example, the total or proportional time spent on an AOI.

In the context of Java program debugging with multiple representations, a modified version of the RFV has been employed in various studies (Romero, Cox, et al., 2002; Romero, du Boulay, Cox, & Lutz, 2003; Romero, du Boulay, Lutz, & Cox, 2003; Romero, Lutz, et al., 2002). In these studies, a software development environment (SDE) was built on top of the RFV and was employed to track visual attention to investigate the coordination strategies of programmers debugging Java programs, who were working with multiple adjacent representations. Several eye-movement-like metrics were derived, to identify superior debugging strategies of participants or to measure the effects of different visualizations on the coordination strategies. For example, a balance in accumulated fixation times between different representations (regions) could reflect a good debugging performance (Romero, du Boulay, Lutz, & Cox, 2003). Another measure derived in the aforementioned studies was the number of switches per minute between the representations of a program. More experienced programmers were found to switch more frequently between the main representations.

Previous research focusing on program debugging and differences between novice and skilled programmers has shown the superiority of expert programmers over novices in terms of domain knowledge, performance, and strategies: Expert programmers found more bugs, found them faster, and tended to spend more time on building a mental model of the problem (Gugerty & Olson, 1986). Experts also (1) are more able to remember specific parts of the source code (Fix, Wiedenbeck, & Scholtz, 1993), (2) focus only on relevant information needed to solve the problem (Koenemann & Robertson, 1991), (3) are not committed to one interpretation, as novices are (Vessey, 1985), and (4) are therefore able to change their strategies as needed.

Figure 1 shows a screen shot of the SDE when the restricted condition of the RFV was turned on. The SDE's interface contains three main areas: The code is in the left window pane, the frame containing a visualization of the program is in the top-right pane, and the output is at the bottom right. In Figure 1, the focused region, as set by the user's mouse, is located over the visualization in the top-right pane.

The RFV has also been employed in the research of shifts of visual attention during integration of text and graphics (Futrelle & Rumshisky, 2001). Because the original version of the RFV was limited to only static stimuli, other researchers tried to improve the RFV idea, using an enhanced version of RFV for usability studies of hyperlinked documents (Tarasewich & Fillion, 2004).

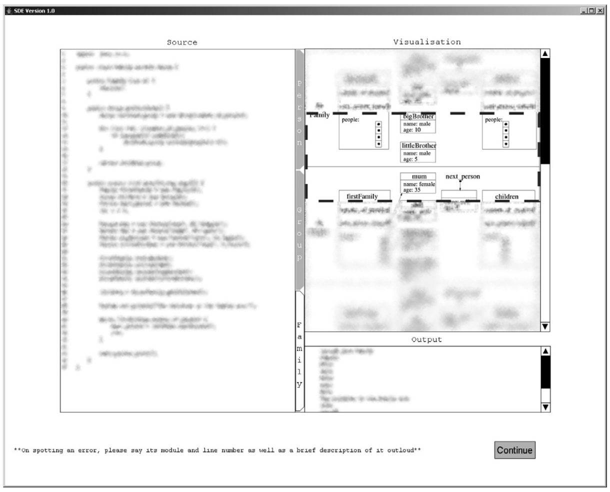
Experiment

We are very interested in the possibilities and limitations of experimental tools, and want to find out whether their use actually interferes with the (otherwise unaltered) behavior of the participants in an experiment. The purposes of the present experiment were (1) to compare the RFV-based measurements of visual attention shifts to the data obtained through eye-movement tracking, and (2) to investigate possible interference of the blurring technique used by the RFV with the debugging strategies and performance. We fully replicated a previous experiment (Romero, Lutz, et al., 2002) that employed the RFV; a remote eye-movement tracker also recorded the visual attention shifts of the participants. The changes in visual attention when the stimuli were blurred and fully focused were analyzed.

METHOD

Participants

A total of 19 participants were recruited from a population of students, researchers, and teachers from the authors' department. One participant withdrew from the experiment prematurely; therefore, the analysis is based on the data recorded from 18 participants. All of the participants had normal or corrected-to-normal vision, according to their own report, and had never taken part in an eyetracking experiment. The average age was 25.3 (*SD* = 4.4) years. Three



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

Figure 1. The software debugging environment built on top of the RFV. The focused area is at the top right on the visualization (dashed line added by the authors).

participants were female. The programming and Java experience of the participants varied: Some had just passed a Java course and had little experience, whereas others were professionals working in programming-related careers. Experience with Java programming, and especially professional background, was chosen as the main factor for classification. The participants were divided into two groups. The less experienced group consisted of 10 programmers, who had an average of 63 (*SD* = 33.1) months of programming experience, 8.13 (*SD* = 6.1) months of which were Java programming. No novice participant had ever worked as a professional programmer. The expert group was formed from the remaining 8 participants; they had a mean programming experience of 96 (*SD* = 24.7) months and a mean Java experience of 16.25 (*SD* = 17.9) months, and all of them, except 1, had professional programming experience.

Materials and Design

The target programs used in this study were identical to those used in Romero, Lutz, et al. (2002). The warm-up program inspected whether a point was inside a rectangle. The first program (the "Family" program) printed out the names of the children of a sample family, and the second program (the "Till" program) counted the cash in a cash-register till, giving subtotals for the different denominations of coins. In their study, Romero, Lutz, et al. (2002) had two versions of the target programs; the main difference between the versions was that the second one was a more sophisticated version of the first one. In our replication, we used the less sophisticated versions of Romero's programs and the graphical functional representations in visualizations.

The two main experimental programs contained four errors each; the warm-up program was seeded with two errors. Following the classification of the errors established in Romero, Cox, et al. (2002), Romero, du Boulay, Cox, and Lutz (2003), Romero, du Boulay, Lutz, and Cox (2003), and Romero, Lutz, et al. (2002), the errors in the target programs could be classified as functional, control-flow, and data-structure errors. There were no syntactical errors in the programs (all of the programs could be compiled), and the participants were notified of this.

We used a mixed design with one within-subjects factor (RFV restricting condition, hereafter referred to as RFV-on/RFV-off) and one between-subjects factor (level of experience), with four dependent variables (number of errors spotted, accumulated fixation time, mean fixation duration, and switching frequency, as measured by an eyetracker). The accumulated fixation time is the total time a participant spent during a session fixating an area of interest (AOI). For an AOI, all of the fixations were summed, and the number was divided by the total fixation count throughout the debugging session, giving the mean fixation duration. The switching frequency refers to the average number of attention switches per minute between each of the AOIs. The mean fixation duration is a measure related to the depth of required processing and therefore to the mental workload during the task (Goldberg & Kotval, 1998). Most of the results were analyzed by performing ANOVA and/or planned paired *t* tests.

Procedure

The experiment was conducted in a quiet laboratory. Each participant was seated in a comfortable chair facing a 17-in. TFT display, at a viewing distance of about 80 cm. Before the experiment, the participants had to successfully go through an automatic eyetracking calibration procedure. After that, the participants read detailed instructions about the experiment and the environment used. Participants debugged three programs. The first warm-up session was performed under the RFV restricted-view (RFV-on) condition so that the participants could become familiar with controlling the focused spot and operating the debugging environment. Then, the two main debugging sessions were performed; one session was performed under the RFV-on condition, and the other session was performed under the RFV-off condition, in which the whole display was presented in focus. The order of the programs and conditions was counterbalanced.

Each session had two phases. First, the specification of the program was displayed. It described the problem the program was supposed to solve and the approach to the solution. Two sample interactions were provided—the desired behavior and the actual behavior of the program. Second, the participants were given 10 min (this limitation was taken from the previous studies) to debug the program; they were instructed to find as many errors as possible and to report them aloud.

Finally, after the debugging session, the participants were informally interviewed.

Apparatus

The SDE used in the previous studies (Romero, Cox, et al., 2002; Romero, du Boulay, Cox, & Lutz, 2003; Romero, du Boulay, Lutz, & Cox, 2003; Romero, Lutz, et al., 2002) was employed for the experiment as a source of stimuli. In these studies and in the present experiment, RFV Version 2.1 was used. The settings of the underlying RFV-based mechanism for blurring the stimuli were the same as in the replicated studies (e.g., Romero, Cox, et al., 2002); the program code, the visualization, and the output were precomputed and static. There were three transition steps between the clear view and the fully blurred stimuli, and all of the regions were rectangular, with widths and heights as follows (from outermost to the fully focused region, in pixels): 800 × 400, 700 × 300, 620 × 240, and 540 × 190. It was not possible to read the source code when it was blurred, and the speed of motion blur was set to 1 pixel per sec. Figure 2 illustrates the level of RFV-blurring in the experiment.

For eyetracking, the remote Tobii ET-1750 (sampling rate of 30 Hz) eyetracker was used. The device is built into the 17-in. display (resolution of 1,280 × 1,024), makes no contact with participants, and contains no movable or audible parts that could possibly interfere with the participants (Figure 3). The eyetracking data were collected throughout the whole experiment; the RFV was able to collect data only in the RFV-on condition. The AOIs were defined to correspond with the three main panels in the SDE window: the code, the visualization, and the output pane.

RESULTS

The RFV as a Visual Attention Tracking Tool

To investigate the ability of the RFV to accurately record the switches in visual attention between the areas of the interface, we compared the number of switches per minute. Six types of switches, between three main areas (code, visualization, and output), were possible, as shown in Figure 4. We compared the switching behavior as measured by the RFV to the number of switches as measured by the eyetracker, and we analyzed the differences with the blurring (RFV-on) and without the blurring (RFV-off). Because the RFV cannot measure any switching in visual attention without having the display blurred, only data from the eyetracker were available for that condition.

Three separate two-way ANOVAs (tool × switch type) were run to compare the conditions and measurement tools. First, the RFV- and eyetracking-based data under the RFV-on condition were compared, to investigate the effects of tool. The main effect of tool on the number of switches was significant [$F(1,17) = 30.23, p < .001$], and the interaction effect between tool and switch type was significant [$F(5,85) = 7.83, p < .001$].

The second comparison included the number of switches as measured by the eyetracker, with and without the blurring condition. The main effect of the blurring on the number of switches was significant [$F(1,17) = 5.90$, $p < .05$], and the interaction between blurring and type of switch was not significant [$F(5,85) = 0.48$, n.s.].

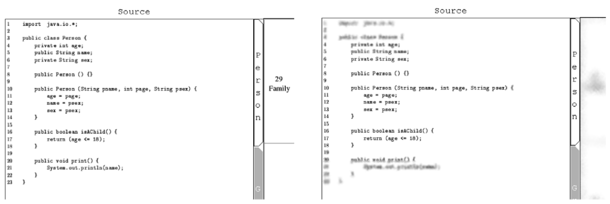
Finally, to compare the tools in their natural settings, we analyzed the differences between the number of switches as measured by the RFV in the RFV-on condition and the number of switches as measured by the eyetracker in the RFV-off condition. The main effect of tool and condition was significant [$F(1,17) = 32.62, p < .001$], and the interaction effect with type of switch was also significant [$F(5,85) = 5.58$, $p < .001$]. For all three comparisons, there was a main effect of switch type on the number of switches [$F(5,85) = 35.37$, $p < .001$; $F(5,85) = 23.94, p < .001$; $F(5,85) = 22.32, p <$ .001, respectively].

The shapes of the amplitudes of the values shown in Figure 4 might suggest that the data obtained by the two tools could be systematically correlated. In other words, the RFV might constantly report lower numbers of switches than the eyetracker, and the relation between these two might be linear. To complement the previous results, we therefore analyzed how the measurements from the two tools and between the two blurring conditions were correlated (Table 1). No significant correlation was found between the measures obtained using the RFV and those recorded by the eyetracker.

Effects of Display Blurring

Since the previous results raised some doubts about the reliability of the RFV technique, the rest of the results, related to the behavior of the experimental groups, were obtained using only the remote eyetracker.

Effects of display blurring on debugging performance. The debugging performance was measured by the number of errors spotted. Under the RFV-on condition, the less experienced group found 2.1 (*SD* = 1.10) errors, on average, and the more experienced group spotted 3.125 (*SD* = 0.84) errors, on average [$t(16) = 2.18, p < .05$]. Under the RFV-off condition, the less experienced group found 2.1 (*SD* = 0.88) errors, on average, and the more experienced group spotted 2.88 (*SD* = 1.13) errors, on average [$t(16) = 1.65$, n.s.]. According to an ANOVA, the effect of the restricted-view condition on the debugging performance was not significant, whereas the effect of



```
import  java.io.*;

public class Person {
    private int age;
    public String name;
    private String sex;

    public Person () {}

    public Person (String pname, int page, String psex) {
        age = page;
        name = psex;
        sex = psex;
    }

    public boolean isAChild() {
        return (age <= 18);
    }

    public void print() {
        System.out.println(name);
    }
}
```

```
[illegible]

[illegible]
    private int age;
    public String name;
    private String sex;

    public Person () {}

    public Person (String pname, int page, String psex) {
        age = page;
        name = psex;
        sex = psex;
    }

    public boolean isAChild() {
        return (age <= 18);
    }

[illegible]
```

Figure 2. Two snapshots detailing the source panel with RFV blurring off (left) and on (right).

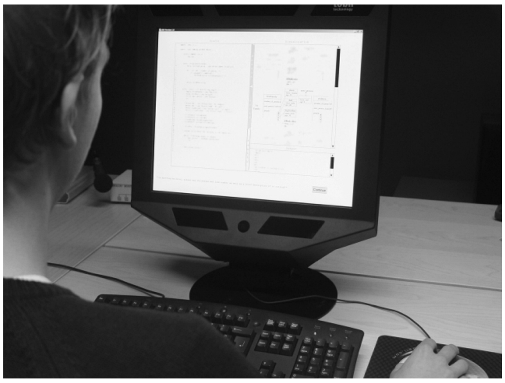


Figure 3. The remote eyetracking device used in the experiment.

experience on the number of bugs found was significant [$F(1,16) = 5.28, p < .05$].

Effects of display blurring on debugging strategies. According to ANOVA, the distribution of relative accumulated fixation time, as measured by the eyetracker, was not affected by the RFV condition for either of the two experimental groups (Figure 5). Novice participants spent, on average, 82% of the total time fixating on the code panel, 14% of the time on visualization, and 4% of the time on the output AOI. For experts, the relative accumulated fixation time followed a distribution of 87%, 10%, and 3%, respectively. Although no effect of experience on the fixation time was found, the effect of AOI was significant [$F(2,32) = 623.07, p < .001$].

Dynamics of attention-switching behavior were measured by the eyetracker as the number of switches between any of the three main representations of the program (the code, the visualization, and the output). Figure 6 presents the results obtained by the eyetracker for the blurred (RFV-on) and fully focused (RFV-off) stimulus. On average, the experts performed 4.93 (*SD* = 1.93) and 8.86 (*SD* = 4.09) switches per minute under the RFV-on and RFV-off conditions, respectively, whereas novices performed 7.00 (*SD* = 2.13) and 7.76 (*SD* = 3.50) switches per minute under the RFV-on and RFV-off conditions, respectively. The effect of RFV condition was significant [$F(1,16) = 7.82, p < .05$], and the interaction between level of experience and RFV condition was significant, $\alpha = .92$ [$F(1,16) = 3.59, p < .08$]. We observed a decrease in the number of switches per minute when the display was blurred, which was significant for experts [$t(7) = 2.53, p < .05$]. Moreover, the average number of switches per minute of novices was significantly correlated under the RFV-on and RFV-off conditions [$r(10) = .642, p = .046$], whereas the same correlation for experts was low and not significant [$r(8) = .068, p = .873$].

The mean fixation duration is often used as a measure of cognitive workload and depth of required processing (Goldberg & Kotval, 1998). We analyzed the mean fixation durations, measured by the eyetracker for the two experimental groups and the two blurring conditions, for the three main areas of the SDE interface and overall (Figure 7). The analysis of the results revealed an effect of RFV condition on mean fixation duration [$F(1,16) = 4.45, p < .051$], and no interaction between level of experience and RFV condition [$F(1,16) = 0.26$, n.s.]. The effect of experience on the mean fixation duration approached significance [$F(1,16) = 3.6, p = .076$]. The mean fixation durations on the three areas were significantly different [$F(2,16) = 10.13, p < .005$]. The planned paired *t* tests revealed that, for experts, the overall mean fixation duration and the mean fixation durations over the code AOI differed significantly between the RFV-on and RFV-off conditions [$t(7) = 2.80$; $t(7) = 2.66$, respectively; all ps $<$ .05]. The overall mean fixation durations of the experts were 308.82 msec (*SD* = 83.95) and 263.09 msec (*SD* = 70.60) under the RFV-on and RFV-off conditions, respectively. For the code panel, the mean fixation durations of the expert group were 312.44 msec (*SD* = 85.69) and 268.23 msec (*SD* = 73.64) under the RFV-on and RFV-off conditions, respectively. For the novice group, there was no significant difference in the fixation durations between the RFV-on and RFV-off conditions, according to the pairwise

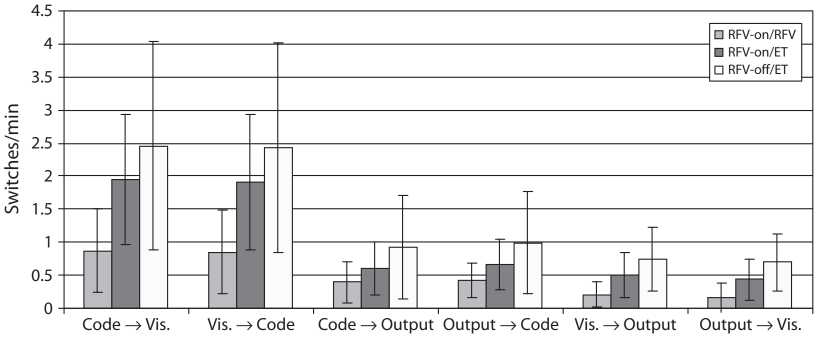


Figure 4. The average number of switches per minute as measured by the RFV and the eyetracker (ET) under the two conditions for all six types of switches.

Table 1
Correlations of Number of Switches per Minute Between RFV-On and RFV-Off Conditions, as Measured by the RFV and Eyetracker (ET), in the Format "Tool (Condition)"

	Code→ Vis.	Vis.→ Code	Code→ Output	Output→ Code	Vis.→ Output	Output→ Vis.
RFV (RFV-on) vs. ET (RFV-on)	.1694	.1465	.1741	.4184	.3716	.0358
ET (RFV-on) vs. ET (RFV-off)	.5578*	.5467*	.3027	.2508	−.3390	−.0690
RFV (RFV-on) vs. ET (RFV-off)	.0588	.1579	.0391	−.0522	−.1610	−.1620

Note—Vis., visualization. $^*p < .05$, two-tailed *t* test.

tests. The overall mean fixation durations of the novices were 396.90 msec (*SD* = 112.29) and 381.44 msec (*SD* = 112.62) under the RFV-on and RFV-off conditions, respectively. No statistically significant difference between them was found, and other measurements of novice mean fixation durations also did not differ significantly under the RFV-on and RFV-off conditions.

Summaries of the interviews. All of the experimental participants were informally interviewed after the debugging session, to investigate their attitudes and opinions about the display blurring. We discovered a general pattern occurring in the statements of subjective attitudes: Novice programmers greatly appreciated the fact that the environment reduced the wealth of visual stimuli, allowing them to concentrate better on extracting the information from the display. Experts, on the other hand, universally disliked the blurring feature of the environment. As an extreme example of negative attitude toward the display blurring, an expert—who declined to participate in the experiment shortly after the warm-up program appeared blurred on the screen—stated in the interview: "I do not want to work with that." Consequently, no data were recorded for this person.

DISCUSSION

This study addressed an important issue: whether tracking of visual attention through eye-movement technology can be substituted by the tracking of a focused spot within an otherwise blurred stimulus. The eyetracker records a location of gaze direction, which is thought to be tightly coupled with the direction of visual attention. The RFV assumes that participants constantly look at the only focused spot within otherwise blurred stimuli. We replicated one of the previous experiments that employed RFV technology (Romero, Lutz, et al., 2002), but we also recorded the gaze data, with the help of a remote eyetracker. We compared the data provided by these two tools and also analyzed the changes in the behavior of participants when the stimulus was presented in a blurred form. We presented several findings related to the performance and behavior data.

The RFV has already been validated previously (Jansen et al., 2003). However, the tasks used in the validation were not very complex and involved only one graphical representation of a problem, in contrast to the program debugging, which involved participants' reasoning with multiple representations of the program. Moreover, the previous validation did not consider the possible effects of the blurring on the mental workload of the participants.

If participants' performance is measured in terms of the number of bugs discovered under the two conditions, then no significant effect of display blurring was found on either of the experimental groups. Overall, the more experienced group performed better, but this was linked with their experience and was an expected result that has also been found by other empirical studies of novice and expert programmers (e.g., Gugerty & Olson, 1986). The performance of the more experienced participants, however, improved slightly under the blurring condition.

In comparison with the eyetracker, the RFV does not seem to accurately track the switching of visual attention of programmers while they are working with multiple representations of a program. The frequencies of visual attention switching as measured by the RFV were always lower in our experiment than were the frequencies measured by the remote eyetracker, and the two streams of data were

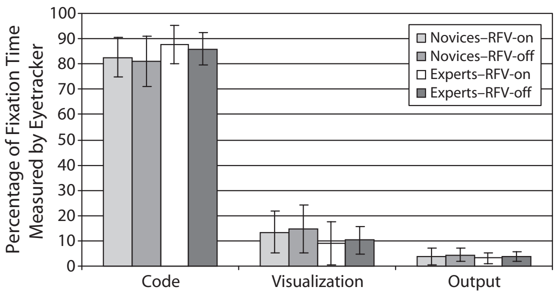


Figure 5. Proportions of accumulated fixation times spent on three areas of interest, as measured by the eyetracker.

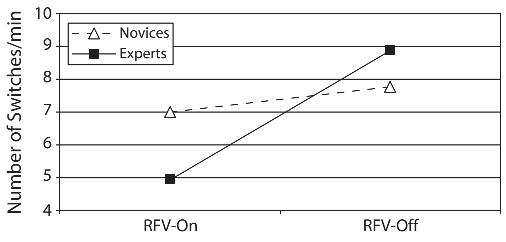


Figure 6. The average number of switches, as measured by the eyetracker.

not correlated under the restrictive blurred-display condition. Moreover, we found that the blurring condition interfered with the debugging strategies of the participants. Therefore, the results from the present experiment do not agree with the previous validation studies of RFV (Jansen et al., 2003). This finding can be explained both from a methodological view and from a theoretical perspective provided by previous studies of programmers.

When targets are reasonably large, gaze has been shown to provide faster ways of interaction than does mouse selection (Sibert & Jacob, 2000; Ware & Mikaelian, 1987). By observing the video recordings taken while the stimuli were blurred (for an example screen shot, see Figure 8), we revealed that it was often the case that participants gazed at locations other than the focused region as defined by the RFV position. Usually, the participants set the focused spot to a certain location and then performed several quick glances away and attention switches without moving the RFV-focused spot. In this way, the focused region was not used as the only location for extracting the information from the environment, but served as a kind of bookmark to the current representation. This behavior seemed to be frequent and intentional, most probably providing participants a memory refresh of the previously visited locations. In those situations, the costs of the required movements of the hand-controlled focus window were not justified, in comparison with the relatively effortless movement of the eyes. The RFV-based measures therefore cannot be equivalent to or estimates of the changes in visual attention, as is also shown by their low correlations to the eyetracking measures. When the display was not restricted, the cost of a visual attention switch decreased further and was rewarded with unblurred information, as is evidenced by the significant increase in switching frequency.

To analyze where the differences between the RFV-on and RFV-off conditions come from, and whether the effects of the blurring interact with the levels of experience, we conducted a more detailed analysis using eye-movement data alone. The results reveal some effects of the blurring condition on gaze behavior. These effects were found to be more serious for more experienced participants, and they materialized on the attention-switching behavior rather than on the distribution of fixation times. Novice programmers used approximately the same strategy regardless of the display blurring, as can be seen from the high correlations in their switching frequency. Experts, on the other hand, when the display was blurred, changed their natural coordination strategies. When the information was easily available, under the RFV-off condition, experts seemed to integrate more information, as indicated by the increased number of switches.

The differences between the fixation durations of expert and novice participants were caused by the differences in levels of experience, and this result supports findings from other studies (e.g., Bednarik, Myller, Sutinen, & Tukiainen, 2005). Although we did not find any effect of the blurring on the fixation duration of novices, the increase in the fixation duration of experts was surprising, at first glance. However, as the experts changed their natural strategies and performed less information integration from the representations when the display was blurred, they had to spend more mental resources to complete the task. As a result, the increased fixation duration reflects the additional mental workload caused by the blurred display: Since experienced programmers are known to form good hypotheses about the comprehended problem and to be more able to remember specific parts of the code (and presumably also of the visualization and output), they had to decide at each moment exactly what information needed to be extracted from the restricted environment. This is in line with findings of Koenemann and Robertson (1991), who showed that experts concentrate only on relevant information while comprehending program code. Our results can be seen as extending this finding to multirepresentational programming environments: When restricted, experts conducted the coordination of the dif-

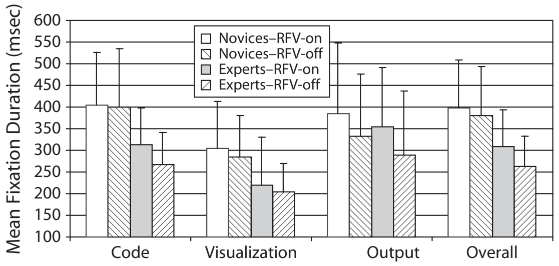
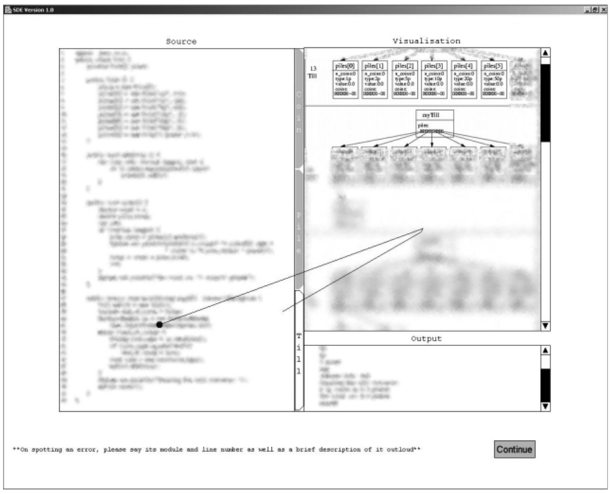


Figure 7. Mean fixation durations over three main areas and overall.



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

Figure 8. A screen shot from interaction, with 1 sec of gaze path and a fixation point superimposed. The focused region is on the top right.

ferent representations mentally rather than visually, yet with an equally good outcome.

We believe that the blurring of the display causes disturbances to the debugging behavior, and these disturbances were more serious for experts than for novices. The results of the interviews can be seen as additional support for this conclusion. Whereas the novices did not consider debugging with a blurring display an unusual task, experts complained about the unnaturalness of the task. However, the performance of the two groups was not influenced, suggesting that experts had to adopt some new, yet equally successful strategies to cope with the restricted view.

The RFV technique is cheap and requires no new equipment to record and analyze the experimental data, although some software efforts are required. However, it produces essentially the same stream of complex data as does eye-movement recording, so it does not simplify the analysis and interpretation of data. On the other hand, the stream of eye-movement data is often highly noisy, and is corrupted by the head movements of the participants or by additional reflections from the eyeglasses. For instance, for an accurate analysis of the switching frequency and total gaze durations, it is necessary that the gaze be constantly available. If the gaze is lost from the view of the eyetracking camera for a certain period, the computation of the eye-movement metrics cannot be accurate. Although eyetracking requires a calibration before the experiment, Jones and Mewhort (2004) have shown that the information filter (focus window) of the RFV (and similar devices) also needs to be calibrated.

Another inherent issue related to RFV-based studies concerns the question of experimental setting. For example, in the study we replicated, one might ask how natural the interaction and debugging strategies are when (1) the stimuli are precomputed, (2) the environment does not allow for modifications of the source code, and (3) the participants have to use an unnatural interface. As seen from the results of this experiment, the blurring does indeed interfere with the natural strategies of participants. Jones and Mewhort (2004) suggested that the size and characteristics of the view in focus affect the information search strategies. In the present experiment, the size of the focused window was equal to those used in previous investigations using the RFV. If the size of the focused window and the level of blurring of the periphery were to be made more restrictive, to prevent the unwanted use of peripheral information, it is highly probable that the participants who rely on peripheral data would have to put more energy into moving the mouse to get the information they need. The resultant behavior would probably be even more disrupted, and the results of such a study would diverge further from natural conditions. However, this question will have to be answered in future studies.

Conclusions

Although the RFV is designed not to replace eyetrackers, but to be an alternative to the expensive technology, the results of this replication study show that some experimental tasks and designs might not fit well for the RFV. For those tasks, the visual attention data recorded using the RFV and eyetracking differ significantly. Moreover, the blurring of the stimuli changes the natural strategies used, depending on experience levels. We provided possible methodological and theoretical explanations of the differences found.

Although the RFV was introduced as a complementary tool for visual attention tracking, these findings make the use of the RFV, as well as the conclusions of the RFV-based experiments on complex problem-solving tasks, questionable.

AUTHOR NOTE

This work was supported by a grant from the Faculty of Science, University of Joensuu. Correspondence concerning this article should be addressed to R. Bednarik, Department of Computer Science, University of Joensuu, P.O. Box 111, 80101 Joensuu, Finland (e-mail: bednarik@cs.joensuu.fi).

REFERENCES

Bednarik, R., Myller, N., Sutinen, E., & Tukiainen, M. (2005). Effects of experience on gaze behavior during program animation. In P. Romero, J. Good, E. Acosta Chaparro, & S. Bryant (Eds.), *Proceedings of the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG '05)* (pp. 49-61). Brighton, U.K.: University of Sussex.

Bednarik, R., & Tukiainen, M. (2004). Visual attention and representation switching in Java program debugging: A study using eye movement tracking. In E. Dunican & T. R. G. Green (Eds.), *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG '04)* (pp. 159-169). Carlow, Ireland: Institute of Technology.

Fix, V., Wiedenbeck, S., & Scholtz, J. (1993). Mental representations of programs by novices and experts. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '93)* (pp. 74-79). New York: ACM Press.

Futrelle, R. P., & Rumshisky, A. (2001). Discourse structure of text–graphics documents. *Proceedings of the 1st International Symposium on Smart Graphics*. Hawthorne, New York: ACM Press.

Goldberg, J. H., & Kotval, X. P. (1998). Eye movement–based evaluation of the computer interface. In S. K. Kumar (Ed.), *Advances in occupational ergonomics and safety* (pp. 529-532). Amsterdam: IOS Press.

Gugerty, L., & Olson, G. M. (1986). Comprehension differences in debugging by skilled and novice programmers. In E. Soloway & S. Iyengar (Eds.), *Empirical studies of programmers: First workshop* (pp. 13-27). Norwood, NJ: Ablex.

Hyönä, J., Lorch, R. F., Jr., & Kaakinen, J. K. (2002). Individual differences in reading to summarize expository text: Evidence from eye fixation patterns. *Journal of Educational Psychology*, **94**, 44-55.

Jansen, A. R., Blackwell, A. F., & Marriott, K. (2003). A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, & Computers*, **35**, 57-69.

Jones, M. N., & Mewhort, D. J. K. (2004). Tracking attention with the focus-window technique: The information filter must be calibrated. *Behavior Research Methods, Instruments, & Computers*, **36**, 270-276.

Just, M. A., & Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology*, **8**, 441-480.

Koenemann, J., & Robertson, S. P. (1991). Expert problem solving strategies for program comprehension. In S. P. Robertson, G. M. Olson, & J. S. Olson (Eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Reaching through technology* (pp. 125-130). New York: ACM Press.

Law, B., Atkins, M. S., Kirkpatrick, A. E., & Lomax, A. J. (2004). Eye gaze patterns differentiate novice and experts in a virtual laparoscopic surgery training environment. *Proceedings of the 2004 Symposium on Eye Tracking Research and Applications* (pp. 41-48). New York: ACM Press.

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

Romero, P., Cox, R., du Boulay, B., & Lutz, R. (2002). Visual attention and representation switching during Java program debugging: A study using the Restricted Focus Viewer. In *Diagrammatic Representation and Inference: Second International Conference, Diagrams 2002, Callaway Gardens, GA, USA. April 18–20, 2002: Proceedings* (Lecture Notes in Computer Science, Vol. 2317, pp. 221-235). Berlin: Springer.

Romero, P., du Boulay, B., Cox, R., & Lutz, R. (2003). Java debugging strategies in multi-representational environments. In M. Petre (Ed.), *Proceedings of the 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG '03)* (pp. 421-434).

Romero, P., du Boulay, B., Lutz, R., & Cox, R. (2003). The effects of graphical and textual visualisations in multi-representational debugging environments. In J. Hosking & P. Cox (Eds.), *Proceedings of the 2003 IEEE Symposia on Human Centric Computing Languages and Environments* (pp. 236-238). Piscataway, NJ: IEEE Computer Society.

Romero, P., Lutz, R., Cox, R., & du Boulay, B. (2002). Coordination of multiple external representations during Java program debugging. In S. Wiedenbeck & M. Petre (Eds.), *Proceedings of the 2002 IEEE Symposia on Human Centric Computing Languages and Environments* (pp. 207-214). Piscataway, NJ: IEEE Computer Society.

Sibert, L. E., & Jacob, R. J. K. (2000). Evaluation of eye gaze interaction. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 281-288). New York: ACM Press.

Tarasewich, P., & Fillion, S. (2004). Discount eye tracking: The Enhanced Restricted Focus Viewer. *Proceedings of the 10th Americas Conference on Information Systems* (pp. 1-9). New York: AMCIS.

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man–Machine Studies*, **23**, 459-494.

Ware, C., & Mikaelian, H. H. (1987). An evaluation of an eye tracker as a device for computer input. *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface (CHI '87)* (pp. 183-188). New York: ACM Press.

(Manuscript received June 16, 2005;
revision accepted for publication January 21, 2006.)

P2.

2

Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: **Effects of Experience on Gaze Behaviour during Program Animation.** In Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05), Brighton, UK, June 28 - July 1, 2005, pp. 49-61

Effects of Experience on Gaze Behavior during Program Animation

Roman Bednarik, Niko Myller, Erkki Sutinen, and Markku Tukiainen

Department of Computer Science, University of Joensuu,
P.O. Box 111, FI-80101 Joensuu, FINLAND
firstname.lastname@cs.joensuu.fi

Abstract. The purpose of program visualization is to illustrate some aspects of the execution of a program. A number of program visualization tools have been developed to support teaching and learning of programming, but only few have been empirically evaluated. Moreover, the dynamics of gaze behavior during program visualization has not been investigated using eye movements and little is known about how program animation is attended by learners with various levels of experience. We report on an empirical study of the gaze behavior during a dynamic program animation. A novice and an intermediate group, a total of 16 participants, used Jeliot 3, a program visualization tool, to comprehend two short Java programs. Referring to previous literature, we hypothesized that the performance as well as the gaze behavior of these two groups would differ. We found statistically significant differences in performance measures and in fixation durations. Other commonly used eye-tracking measures, the fixation count and the number of attention switches per minute, seem to be insensitive to the level of experience. Based on the results, we propose further directions of the research into gaze behavior during program visualization.

1 Introduction

Program visualization is used to illustrate visually the run-time behavior of computer programs. These systems can be utilized, for example, in programming courses to support teaching of programming concepts to novice programmers. Jeliot 3 is an interactive program visualization system that automatically visualizes data and control flows of Java programs. It has been successfully used in classroom settings to teach programming to high school students [1].

Although several program visualization tools exist, only few have been evaluated and little knowledge is available about the aspects of gaze behavior during a dynamic program visualization. It is not clear how different users attend the animation and what cognitive efforts they have to exercise in order to comprehend the dynamic visualization. Therefore, in order to improve program visualization systems to fit their users best, it is a crucial issue to investigate the visual attention paths of users while visualizing a program. If a purpose of program visualization is to support the novices in their understanding, it is reasonable to study how their behaviors differ from the behaviors of intermediates. In other domains, eye-movement tracking has been successfully applied to investigate the gaze patterns of participants while performing their tasks. However, no eye-movement based analysis of the gaze behavior during a dynamic program visualization has been conducted yet.

We report on an initial study in which we have employed a remote eye tracker to measure the gaze behavior of programmers during program comprehension facilitated by an animation tool, Jeliot 3.

The rest of the paper is arranged as follows. In Section 2, we review some related work in eye tracking research and program visualization, and Jeliot 3 is introduced. The experiment and results are described in Sections 3 and 4, respectively, and discussed in Section 5. Conclusions and future work are presented in Section 6.

2 Related Work

2.1 Eye Tracking

Humans move their eyes in order to bring an inspected object or a portion of it onto fovea, the high-resolution area of retina. This way the visual attention is closely linked with the direction of the eye-gaze, and most of the time it is also diverted to the point of visual inspection. Following this assumption, if we can track the movements of eyes, we can also get insights into and investigate the path and focus of attention during a task such as program comprehension. Furthermore, knowing which objects have been visually inspected and in which order and context, we can attempt to infer what cognitive processes were involved to perform the task related to these objects.

Eye tracker is a device that records eye movements. Most of the current eye trackers use infrared light emitters and video image analysis of the corneal reflections and pupil center to relate them to the direction of gaze. Typically, the accuracy of current eye trackers ranges around 1 degree, while the data is sampled at rates of 50–500Hz. Current eye trackers are relatively cheap and able to reliably and unobtrusively collect gaze data.

From the signal obtained from an eye tracker, two most important types of eye movements are usually identified: saccades and fixations [2]. *Saccades* are rapid ballistic movements of eyes that are executed to reposition the eyes from one location of attention to another one. A single saccade can last between 30 and 120 ms, can span over 1 to 40 degrees of visual angle [2], with velocities ranging up to 500 degrees per second [3]. No visual information is extracted during a saccade, a phenomena called saccadic suppression [4]. *Fixations* are eye movements stabilizing the image of an object on the retina. Typical fixation duration ranges between 200–300 ms [3]. It is assumed that during the period of a single fixation the information is extracted, decoded, and interpreted. The fixation duration can be therefore thought to be related with a required processing to extract and interpret the information [5, 6]. An accurate measurement and analysis of eye movements in terms of saccades and fixations provide researchers with the details of cognitive processing and related visual attention allocation within a performed task. For instance, the fixation count or sum of fixation durations on a certain element can be related to the importance of the element. In the context of program visualization interfaces, the relative fixation count measure can correspond with the relative importance of a representation (e.g. a code or a state diagram) of a program.

It is a well-known fact that eye movement patterns of experts and novices differ. Previous eye movement studies in other domains than program visualization have shown, for instance, that (1) search strategies differ between novice and expert radiologists [7], (2) expert-pilots' eye movement patterns were better defined and the dwell times were significantly shorter than those of novices [8]. A common denominator in these and other reports is that domain knowledge and experience of participants seem to be the main factors influencing not only the performance, but also the related gaze behavior.

Visual attention tracking during program comprehension has been previously studied by Crosby and Stelovsky [9]. They used an eye tracker to discover the relationship between cognitive styles and individual differences, and code-reading patterns. In their study, novices and experts were eye tracked during an algorithm comprehension. However, only one representation of program was used (the code) and the focus of the research was mainly on the critical, but surface features of code, not on the behavior during a dynamic program visualization.

In the direction of investigating issues such as visual attention switching or a multiple-representation use during program comprehension or debugging, previous studies involved only a static precomputed stimuli and the analysis was based on a recording of mouse movements over a blurred interface [10, 11]. The validity of such an approach was shown to be questionable [12, 13]. To our knowledge, no eye movement based analysis of behavior during program animation has been conducted yet. This is certainly surprising, considering the importance of knowledge how the visual attention and cognitive processes involved in program comprehension are influenced by program animation.

2.2 Program Visualization

A number of program visualization systems have been developed over the previous years to teach programming or to visually debug programs. Here we will briefly review those systems that in some aspects are similar to Jeliot, the program visualization tool employed in the present experiment.

Javavis [14] is a tool that visualizes automatically the runtime behavior of the Java programs. It shows changes in the state of the program during execution using animated UML-like object and sequence diagrams. DDD [15], a debugging front-end, uses diagrams to illustrate the references between data structures during program execution. The diagram can be seen as graphs where nodes are the separate data structures (e.g. struct in C) and vertices are the references between them. The DDD does not explicitly visualize the control flow of the program. Jive [16] uses a similar approach to Javavis and DDD to visualize the program state using diagrams. The references, primitive values and variables are visualized similarly in Jeliot 3 and these systems. However, only Javavis visualizes control flow, but in less detail compared to Jeliot 3.

PlanAni [17] is a program visualization system that illustrates the data flow of a program during its execution. The use of variables in different purposes is illustrated through the roles of variables. The expression evaluation and control flow are also visualized. Currently, the animations must be programmed beforehand by an instructor and the visualization of object-oriented concepts is not supported. The organization of the user interface in PlanAni is similar to Jeliot. However, Jeliot does not visualize the roles of variables as PlanAni and PlanAni does not visualize the control flow.

2.3 Jeliot 3

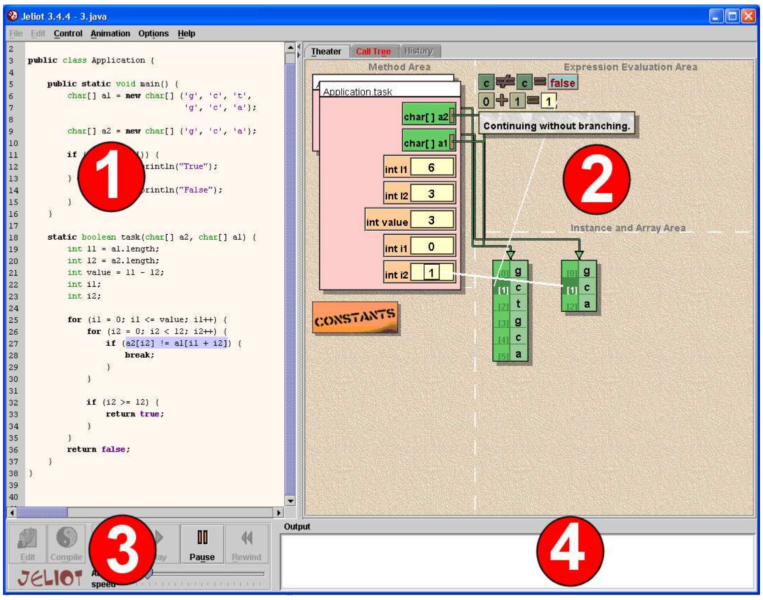
Moreno at al. [18] have developed a program visualization system, called Jeliot 3. Its predecessor, Jeliot 2000, has been successfully used to improve the teaching of introductory programming helping the novices to acquire vocabulary to explain programming structures and concepts [1]. Jeliot 3 retains the novice-oriented GUI and animation display of Jeliot 2000. Jeliot 3 introduced a new design in order to make the system extensible and to allow for adding new features into the visualization. It visualizes automatically the execution of user-written Java programs by illustrating the data and control flow and object-oriented features of the program. Jeliot 3 can visualize a large subset of novice-level Java programs (see `http://cs.joensuu.fi/jeliot/`). The user interface of Jeliot 3 is shown in Figure 1.

The interface consists of four discrete areas. A code editor on the left hand side shows the program code, and during program visualization, the currently executed statement or expression is highlighted. A control panel in the bottom left corner is used to control the animation with VCR-like buttons. The largest area of the user interface of Jeliot is occupied by the visualization view showing the execution state of the program on the right hand side of the window. Visualization consists of method frames, local variables, expression evaluation, static variables, objects and arrays. Finally, an output console lies in the bottom right corner of the window, showing the output of the executed program. To sum it up, Jeliot provides four different areas of interest to the user: code view, animation view, control panel, and output console. Moreover, animation view is further divided into four different areas of interest: method, expression evaluation, constant, and object and array areas. Furthermore, there are separate specialized visualizations where only the call tree of the program or the execution history are shown.

In a typical session with Jeliot, a user either writes or loads a previously stored program. User can compile the program through the user interface of Jeliot. When the program is compiled, a visualization view, where the user can see the animation of the program execution, is opened. Jeliot shows the execution either step by step or continuously. User can control the speed of the animation and stop or rewind the animation at any point. User can select the current visualization with the tabs on top of the visualization view.

3 Experiment

The present research investigates the differences in the gaze behavior during program animation of participants with different levels of programming experience. Based on the results from available literature, our hypothesis was that the performance and gaze behavior of novices and intermediates differ during the program animation. In other words, our aim was to answer the question, whether intermediates and novices pay attention to the animation in a similar or different way. Our hypothesis is not surprising, since we naturally assume that a different level of experience shall result into a different gaze behavior and performance, as it has been found in other domains. More experienced programmers are expected to form better hypotheses about the problem and this knowledge should guide them to use the available representations in a distinct



```
public class Application {

    public static void main() {
        char[] a1 = new char[] {'g', 'c', 't',
                                'g', 'c', 'a'};

        char[] a2 = new char[] {'g', 'c', 'a'};

        if (          1)) {
                    println("True");
        }
                    println("False");
        }
    }

    static boolean task(char[] a2, char[] a1) {
        int l1 = a1.length;
        int l2 = a2.length;
        int value = l1 - l2;
        int i1;
        int i2;

        for (i1 = 0; i1 <= value; i1++) {
            for (i2 = 0; i2 < l2; i2++) {
                if (a2[i2] != a1[i1 + i2]) {
                    break;
                }
            }

            if (i2 >= l2) {
                return true;
            }
        }
        return false;
    }
}
```

Fig. 1. User interface of Jeliot 3. Area 1 is code editor, area 2 is animation frame, area 3 is control panel and area 4 is output console.

way, compared to novices. We had a further assumption that novices would rely more on the visualization than code and the other way around for intermediates.

To validate these hypotheses, we conducted an empirical experiment where we used a remote eye-tracker to record the gaze behavior of the participants during program comprehension task aided by an animation. Two groups of participants with different level of experience used Jeliot 3 to comprehend three short Java programs while their eye movements were simultaneously tracked.

3.1 Method

We used a between-subject design with experience (novice or intermediates) as the factor. The depended variables were: relative fixation count over the areas of interest, number of switches per minute and mean fixation duration over the areas of interest and in overall. The fixation count is a measure related to the level of participant's interest in an area. The number of switches per minute is a measure of attention allocation dynamics. The mean fixation duration is associated with the depth of processing required to understand an attended element. Only the gaze data during the program animation were used in this analysis because that is the only time when all the representations were available concurrently and the selection of the attended representation would make a difference in understanding the program. Most of the analysis was carried out using ANOVA and planned comparisons based on t-test.

3.2 Participants

Eighteen participants were recruited from high-school students attending a university-level programming course, undergraduate and graduate computer science students from local university. Due to technical problems with the eye tracking, data from two participants had to be discarded. Therefore, the results are based on the data collected from 16 subjects (13 male, 3 female). Participants were divided into two groups according to their level of programming experience. Participants with less than 24 months of programming experience were regarded as novices and above 24 months as intermediates. The characteristics of the two groups are presented in Table 1. Groups' mean values for programming experience (in months) and Java experience (in months) and counts for previous experience with Jeliot 3 (yes=1, no=0) and previous experience as professional programmer (yes=1, no=0) are shown. Standard deviations are shown in parentheses.

Table 1. Characteristics of the groups. * marks a significant difference between groups in two-tailed t-test (interval values) or χ^2-test (nominal values) with $p < 0.05$

Experience level	Count	Prog. exp.*	Java exp.*	Jeliot exp.	Prof. exp.
Novices	8	12.8 months (6.9)	6.4 months (4.6)	3	1
Intermediates	8	85.5 months (56.4)	19.8 months (15.0)	2	1

3.3 Materials and Apparatus

Three short Java programs, factorial computation, recursive binary search, and naïve string matching were presented to the participants. The lengths of the programs in lines of code were 15, 34, and 38 respectively. Each of the programs generated only one line of output and did not require any user input. The names of methods and variables were altered so that the recognition of a program based on these surface features would be difficult.

In our study, we used an adapted version of Jeliot 3 which logged all the user actions and all the changes in the visualization of the programs to be compared with the eye tracking data. However, this material is not used in this analysis. The specialized visualizations, the execution history and the call tree, were disabled to avoid problems in interpreting the gaze behavior.

The remote Tobii ET-1750 (sampling rate 50Hz) eye tracker making no contact with participants was used to track eye movements; the eye tracker is built into a TFT panel so no moving part is visible and no sound can be heard during the recording. Only a computer mouse was available during the experiment to interact with the tool. The interaction protocols (such as mouse clicks) were collected for all the target programs, and audio and video were recorded for a whole session. Fixations shorter than 100 ms were disregarded from analysis. We have defined four main areas of interest matching the four main areas in the Jeliot interface: the code, the animation, the control, and the output area. Figure 2 illustrates the experimental settings used in the study.

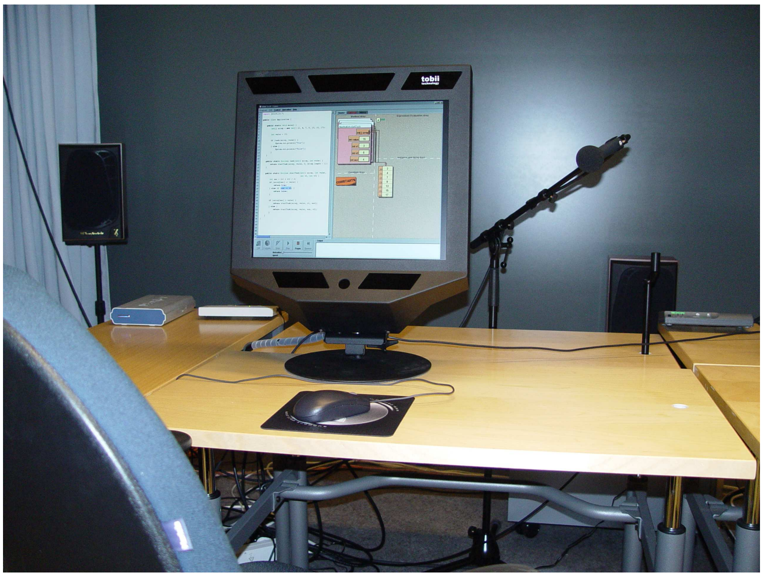


Fig. 2. Experimental settings.

3.4 Procedure and Design

The experiment was conducted in a quiet usability lab. Participants were seated in an ordinary office chair, near the experimenter, and facing a 17" TFT display. Every participant then passed an automatic eye-tracking calibration. During the calibration procedure, a participant had to follow sixteen shrinking points appearing one by one across the screen. If needed, the calibration was repeated in order to achieve the highest possible accuracy.

After a successful calibration, participants performed three sessions, each consisting of a comprehension phase using Jeliot 3 and a program summary writing phase. Participants were instructed to comprehend the program as well as possible and they could interact with Jeliot as they found it necessary. The target programs contained no errors and were always preloaded into Jeliot and compiled. The duration of a session was not limited.

The first program was factorial computation and it was used as a warm-up and the resulting data were discarded. The order of the two actual comprehension tasks was randomized so that half of the participants started with the recursive binary search and other half with naïve string matching.

4 Results

4.1 Completion and Animation Times

Mean completion times for the comprehension phase were 17.6 minutes (SD= 10.0) for novices, and 9.8 minutes (SD=2.6) for intermediates; the difference was statistically significant according to a two-tailed t-test $(t(7) = 2.48, p < .05)$. From that time, novices spent on average 85.4% (SD=9.6) animating the program whereas intermediates spent 52.9% (SD=20.0) of their time to animation; the difference was statistically significant according to the two-tailed t-test $(t(7) = 5.38, p < .01)$.

4.2 Fixation count distribution

Figure 3 shows a relative fixation count distribution over the areas of interest during the animation. Both groups spent most of the viewing time fixating the animation area, 57.4% (SD=11.9) novices, and 54.8% (SD=15.2) intermediates, of all fixations during the program animation. Next, 39.4% (SD=11.2) and 43.3% (SD=14.5), novices and intermediates, respectively, of all fixations was paid to the code area. No significant effect of experience on the distribution of fixations was found, without any interaction between the area of interest and experience. The fixation count has significantly differed between all four areas of interest, $F(3, 42) = 105.75, p < .001$. The planned comparison revealed a significant difference in the fixation count between the two most attended areas, the code and the animation $(t(15) = 2.29, p < .05)$.

4.3 Switching Behavior

Figure 4 illustrates the switching behavior as expressed by the number of switches per minute between the different areas of interest. The average number of switches per minute was 30.15 (SD=10.66) and 27.57 (SD=8.04) for novices and intermediates, respectively. The analysis of the effect of experience on the switching behavior discovered no significant change in the number of switches per minute, $F(1, 14) = 0.004, ns$. The switch between the code and the animation areas was far most common, $F(5, 70) = 145.25, p < .001$. Finally, the interaction effect between type of switch and experience was not significant, $F(5, 70) = 0.421, ns$.

4.4 Fixation Durations

Figure 5 shows the mean fixation durations during animation for the four main areas of interest and the overall mean fixation duration. These have been computed as a sum of durations of all fixations landing at an area of interest divided by number of the fixations. Since the programs did not generate an extensive output, some of the participants were not gazing to this area of interest. For the analysis, the missing values were replaced by the mean value of a group.

The overall mean fixation duration was 406.49 ms (SD=81.40) and 297.26 ms (SD=80.52) for novice and intermediate group, respectively. The effect of area of interest on the mean fixation duration was nearly significant, $F(3, 42) = 2.79, p = .052$.

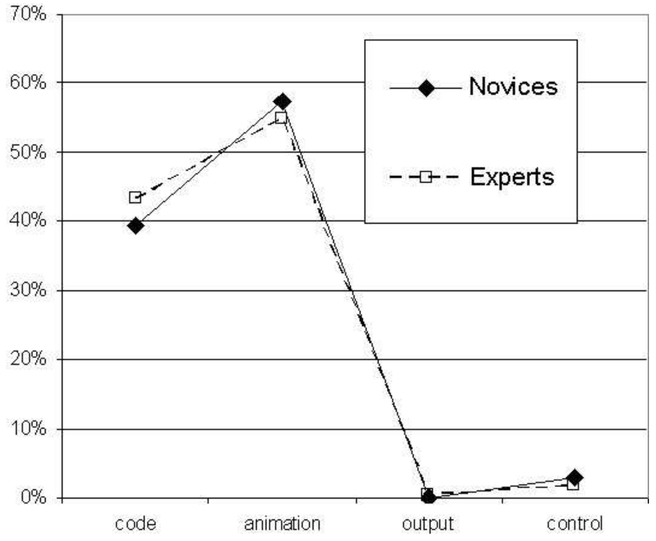


Fig. 3. Relative fixation count distribution during animation.

We also found an interaction between the fixation durations on the areas of interest and the level of experience, $F(3, 42) = 2.87, p = .048$. The effect of experience on the mean fixation durations was significant, $F(1, 14) = 8.98, p = .01$. Moreover, the effect of experience on overall fixation duration, $F(1, 14) = 7.16, p = .018$, was also significant.

5 Discussion

Intermediates completed the comprehension phase much faster than novices. Intermediates also spent significantly less time animating the programs which was in agreement with the hypothesis that intermediates would concentrate more on the code reading. This happened, however, only before they began and after they stopped visualizing the program. Both times can be kept as measures of performance. The initial code-reading episodes could have affected the behavior of the intermediates during the program animation compared to novices. Sajaniemi and Kuittinen [17] reported that during exercise sessions, students using PlanAni did not pay attention to the program code as much as to the visualization. Our results agree with this observation. Although both areas were attended with high fixation counts, it was more common to use the visualization than the code area during program animation, in our study, regardless the experience.

Analysis of the comprehension summaries have been done elsewhere in Bednarik et al. [19] with the program summary analysis by Good and Brna [20]. In this analysis, the summaries of intermediate subjects were found to be slightly better in the quality, but

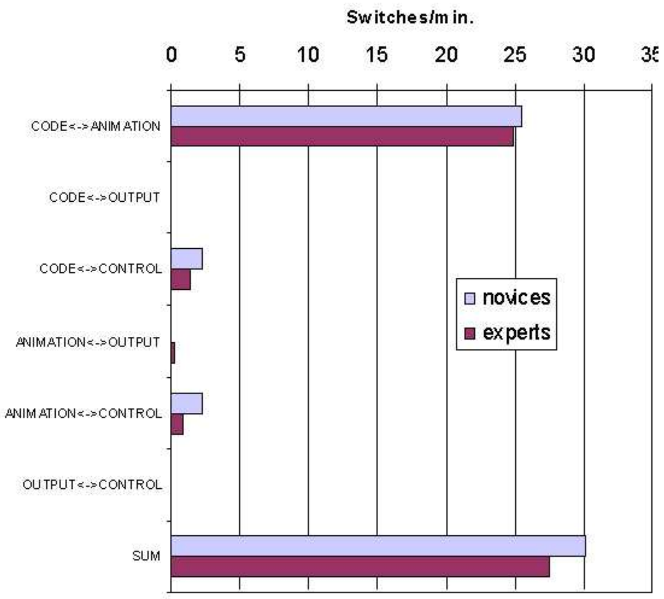


Fig. 4. Number of switches per minute between the main areas of interest.

there were no statistically significant differences found. Intermediates used higher level of abstraction than novices but again there were no statistically significant differences.

The results of this experiment related to the gaze behavior during program animation show that the relative fixation counts and the switching behavior between the areas defined in this study are insensitive to the level of experience. The distribution of fixations between code and animation was slightly more balanced for more experienced participants, but did not significantly differ from the distribution of novice fixations. With respect to these measures, we have to reject our hypothesis. Most of the animation time was spent on viewing the visualization part of the Jeliot interface.

The switches between code and animation areas were the far most common during the animation and therefore the sum of all switches is mostly composed by this type of switch. The code-control and animation-control switches were higher for novices. This is probably due to the fact that novices were interacting more with the tool during the animation than intermediates and therefore attending the control panel more often [19]. In terms of the total number of switches per minute, the two groups exhibited about the same behavior.

With respect to previous eye movement studies investigating the relationship between gaze behavior and expertise, these result are rather surprising. Several factors could, however, explain the results. One explanation seems to be that the features of animation attract equally novice and intermediate programmers to attend the animation in similar patterns. The visualization environment restricts the access to the elements of the graphical representation to only a short period of time, therefore the effects of

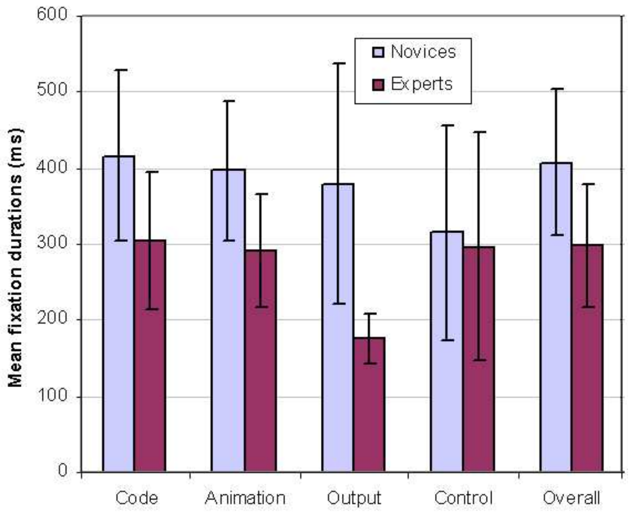


Fig. 5. Mean fixation duration during animation.

experience cannot materialize in the gaze measures used in this experiment. We also believe that more accurate measures have to be developed to reveal the differences between these two groups. For example, we could measure the disassociation between the current animation step and the gaze of the subject. Another possibility for not observing differences in the gaze behavior could be the number of subjects involved in the study and this will be taken into the consideration in the further studies. Finally, the gap between the skills of the two groups involved in this experiment might not be big enough to yield statistically significant differences in gaze behavior during animation.

Despite not finding differences in fixation count distribution and switching behavior, we did find a significant effect of experience on the mean fixation duration. For all the main areas (except for the control area) of the display and in overall, the mean fixation duration of intermediates was shorter than that of novices. This supports the results from previous studies and could be explained by at least two facts or a combination of both. One possibility is that, during the animation, intermediates might have an advantage of already formed hypothesis about the visualized problem. This hypothesis would be formed during the initial code reading before animating the program. The second explanation could be the available domain-knowledge and programming experience of the intermediates which would enable them to interpret the animation faster. From the mean fixation duration over the control panel, we can observe that novices and intermediates alike needed about the same time (300 ms) to decide what buttons they are going to use in order to control the flow on the ongoing animation.

Altogether, these findings could indicate that a difference in the programming experience can be seen in the mental efforts paid while attending the animation, while it does not affect the general patterns how the animation is attended. Both groups attend the suggested attention loci in about same way, but the more experienced programmers extract the information faster and, most probably, are therefore able to pay attention to the surrounding context. When a consecutive attention switch is suggested by the animation, both groups will follow it and thus exhibit similar switching behavior.

6 Conclusion and Further Work

We have conducted an empirical experiment to discover the aspects of gaze behavior during the dynamic program visualization. We employed a non-intrusive remote eye tracking equipment to record the eye movements of programmers with various level of experience. Our results, in terms of the attention switches between different program representations and the distribution of fixations, show no difference in the gaze behavior between novice and intermediate group of programmers during program animation. In other words, the focus of visual attention seems to be distributed in time and space evenly regardless of the experience in programming. When the level of processing required to attend the animation is measured as a mean duration of fixations over the main areas of interest and in overall, our results show that novice programmers spend significantly more time on extracting the features of animated concepts. We propose this difference to be linked to the experience level and with a pre-established model of the algorithm being animated. The performance measures seem to support this hypothesis.

Our initial experiment provides a take-off mark for further studies investigating gaze behavior related to the dynamic program visualizations. Several directions for future research can be taken. Based on the general, macro-level patterns presented in this paper, we aim to deconstruct the behavior into more micro-level sequences. Between our next aims belong to investigate the effects of the discrete animation elements on the gaze behavior as well as the changes in the behavior in a course of time. Among the questions raised by the present study belong, what kind of suggested switches are consumed during the animation and whether the decision differs given the level of experience.

To answer the questions, we plan to develop a methodological framework for a reliable application of eye-movement tracking in the context of program visualization. These studies shall provide us with a deeper understanding about the cognitive processes involved in program comprehension during program visualization.

Acknowledgments

We would like to thank all participants for taking part in this study. We acknowledge Andrés Moreno for a help with preparation of this study.

References

1. Ben-Bassat Levy, R., Ben-Ari, M., Uronen, P.A.: The Jeliot 2000 program animation system. Computers & Education **40** (2003) 15–21
2. Sibert, L.E., Jacob, R.J.K.: Evaluation of eye gaze interaction. In: CHI 2000, ACM Press (2000) 281–288

3. Rayner, K.: Eye movements in reading and information processing: 20 years of research. Psychological Bulletin **124** (1998) 372–422
4. Matin, E.: Saccadic suppression: a review and an analysis. Psychological Bulletin **81** (1974) 889–917
5. Carpenter, P.A., Just, M.A.: Eye fixations during mental rotation. In Senders, J.W., Fisher, D.E., Monty, R.A., eds.: Eye movements and the higher psychological functions. Erlbaum, Hillsdale, NJ (1997) 115–133
6. Goldberg, J.H., Kotval, X.P.: Eye Movement-Based Evaluation of the Computer Interface. In Kumar, S.K., ed.: Advances in Occupational Ergonomics and Safety. IOS Press, Amsterdam (1998) 529–532
7. Nodine, C., Mello-Thoms, C.: The nature of expertise in radiology. In Beutel, J., Kundel, H., Metter, R.V., eds.: Handbook of Medical Imaging. SPIE Press (2000)
8. Kasarskis, P., Stehwien, J., Hickox, J., Aretz, A., Wickens, C.: Comparison of expert and novice scan behaviors during VFR flight. In: The 11th International Symposium on Aviation Psychology. (2001)
9. Crosby, M., Stelovsky, J.: Subject Differences in the Reading of Computer Algorithms. In Salvendy, G., Smith, M.J., eds.: Designing and Using Human-Computer Interfaces and Knowledge-Based Systems. Elsevier (1989) 137–144
10. Romero, P., du Boulay, B., Cox, R., Lutz, R.: Java debugging strategies in multi-representational environments. In: The 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG'03). (2003) 421–434
11. Romero, P., Lutz, R., Cox, R., du Boulay, B.: Co-ordination of multiple external representations during Java program debugging. In: Empirical Studies of Programmers symposium of the IEEE Human Centric Computing Languages and Environments Symposia, Arlington, VA (2002) 207–214
12. Bednarik, R., Tukiainen, M.: Visual attention tracking during program debugging. In: NordiCHI'04, ACM Press (2004) 331–334
13. Bednarik, R., Tukiainen, M.: Effects of display blurring on the behavior of novices and experts during program debugging. In: CHI '05: CHI '05 extended abstracts on Human factors in computing systems, ACM Press (2005) 1204–1207
14. Oechsle, R., Schmitt, T.: JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). In Diehl, S., ed.: Software Visualization. Volume 2269 of Lecture Notes in Computer Science., Springer-Verlag (2002) 176–190
15. Zeller, A., Lütkehaus, D.: DDD — A Free Graphical Front-End for UNIX Debuggers. ACM SIGPLAN Notices **31** (1996) 22–27
16. Gestwicki, P., Jayaraman, B.: Interactive visualization of Java programs. In: IEEE Symposia on Human Centric Computing Languages and Environments. (2002) 226–235
17. Sajaniemi, J., Kuittinen, M.: Program animation based on the roles of variables. In: ACM symposium on Software visualization, ACM Press (2003) 7–16
18. Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M.: Visualizing Programs with Jeliot 3. In: Advanced Visual Interfaces (AVI 2004). (2004) 373–376
19. Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: Analyzing Individual Differences in Program Comprehension with Rich-Data Capture. Submitted (2005)
20. Good, J., Brna, P.: Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. International Journal of Human-Computer Studies **61** (2004) 169–185

P3.

3

Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M.: **Analyzing Individual Differences in Program Comprehension.** Technology, Instruction, Cognition and Learning (TICL), 3 (3-4), special issue on modeling and simulation, 2006, pp. 205-232

Tech., Inst., Cognition and Learning, Vol. 3, pp. 205-232
Reprints available directly from the publisher
Photocopying permitted by license only

Analyzing Individual Differences in Program Comprehension

ROMAN BEDNARIK*, NIKO MYLLER, ERKKI SUTINEN
AND MARKKU TUKIAINEN

Department of Computer Science
University of Joensuu
Joensuu, Finland

Programming is a complex problem-solving domain often involving many dependent entities which may be even hidden or latent. Novice programmers have little knowledge about program execution and may see it as an abstract and non-deterministic process. To support novices, Jeliot was developed to visualize program execution, and thus help in specifying viable program development models. This paper reports on an empirical experiment in program comprehension where 16 subjects used Jeliot to comprehend two Java programs. The experiment focused on how the experience level and complexity of the program affected (a) the patterns of interaction with the tool, (b) the gaze behavior, (c) the use of visualization, and (d) the cognitive processes related to program comprehension. This was done by investigating the protocols obtained from an eye-tracker, interaction logging, and comprehension summaries. An interaction between experience and behavior was found. Experts read the whole code first, constructed a hypothesis, and tested it against the animation. Novice programmers did not read the code first. They animated the program directly, and replayed the animation several times focusing on the difficult sections. The results reveal the potentials of gaze as an additional modality in an adaptive tool for program visualization.

Keywords: Complex systems; Empirical programmer studies; Eye tracking; Program comprehension; Program visualization.

*Corresponding author: Tel: +358 13 251 7977; Fax: +358 13 251 7955; Email: bednarik@cs.joensuu.fi

INTRODUCTION

Programming is a complex and cognitively demanding task due to the multiple interrelated components, tradeoff decisions, and performance requirements that concern the whole process (Detienne, 2002; Hoc, 1990). One of the central parts in the processes of programming, such as creating, maintaining and modifying a software product, is program comprehension. Therefore, the ability to comprehend computer programs is essential and should be learned and supported with proper tools. In order to aid the process of program comprehension and its learning, several tools have already been developed. However, little is known about the user interaction with the tools and about the effects of these systems on the program comprehension and the underlying cognitive processing.

The present study addresses the questions of whether and how a dynamic visualization of program is used during comprehension processes, and how the experience of programmers and complexity of the target programs are reflected in eye-movement patterns and in patterns of interaction. Knowledge of these aspects may help us to create better and more personalized tools and methods for aiding program comprehension.

The complexity of programming

A complex system consists of a large number of components whose interrelations are difficult if not impossible to trace. We can call the complexity of these systems *substantial*. A computer-aided system is supposed to clarify these interrelations and help a human to understand the dynamics of the system, most often by the means of simplifying the interrelations and visualizing the system.

In a sequential program, the dependencies between its structures are – unlike those of a typical complex system – well defined and need no simplification or reduction. The value of a certain variable depends on the predetermined sequence of statements, which make use of other data structures of the program.

However, the complexity lies in the human process of comprehending a program, especially at the novice level. The learner needs to take into account diverse aspects of a program, such as its I/O, control flow, data management, and memory allocation, and to grasp all this information simultaneously makes understanding difficult. Individual preferences, like appropriate representation, complicate things even further: if a learner is exposed to a visualization of, for instance, a variable and its contents that s/he has difficulty to interpret, the representation may mislead her/him even more. Thus, we can characterize the complexity of a sequential program as *cognitive*.

At the same time, depending on the values of the inputs, even a short piece of code might have several different manifestations. This is particularly apparent in the case of short string algorithms, like in the variations of Knuth-Morris-Pratt or Boyer-Moore string searching algorithms. The call tree of a simple recursive program, for example, one operating on a binary search tree, might also result in complex-to-understand structures.

To summarize, the complexity related to programming and its learning is more cognitive than that in a conventional complex system where the complexity is substantial. Therefore, the fundamental challenge of any system that helps a learner to understand or, as a programmer, to control the internal dynamics or operations of a sequential program, is to lessen the cognitive load involved in elaborating the program. This means that the learner should get closer to the actual, often relatively simple, idea of the program. One approach to make the inherent simplicity of any program understandable to a novice programmer is to clarify its operations by visualization. Jeliot, a tool used in the present study, is one of several solutions to this challenge.

Currently, there are many visualization engines or systems available. Most of them have also been evaluated, at least partly; and there are even meta-studies, like that of Hundhausen (2002). However, careful analyses of how a user browses an animation are still mostly lacking. These analyses are essential for developing the visualization environments further, so that they can help the learner to focus on a program's essentials as efficiently as possible. In other words, future systems should uncover the cognitive complexity and give way for the substantial simplicity of the program.

It is worth noting that there are also programs which behave like a complex system and could hence be categorized as substantially complex systems. For example, it is not possible to predict the next step of a concurrent program; the same applies to randomized algorithms, as well as to multithreaded or event-driven programs. However, tools for these kinds of environments are not discussed in the current study.

Program comprehension

A number of studies have been carried out in the field of program comprehension. The theories of program comprehension can be divided roughly into three categories: bottom-up, top-down and mixed models.

The *bottom-up* model of program comprehension was proposed by Shneiderman and Mayers (1979), and Basili and Mills (1982). Pennington (1987) reported that programmers approached a comprehended program in a bottom-up manner from the control structures to the functional structure of the program. However, this model has not received strong support as a comprehensive model for describing the comprehension process, but it is often incorporated as a part of the model.

Brooks (1983) presented a model in which the program was comprehended in a *top-down* manner. The central idea of the theory was that programmers generate hypotheses about the code, using their programming and domain knowledge, and try to verify them. Brooks treated the bottom-up comprehension process as a degenerated special case of the top-down strategy. Letovsky (1986) proposed a similar model and verified the model in an empirical experiment. The conclusion was that the comprehension process is guided by the hypotheses. However, Letovsky also recognized the importance of the bottom-up approach in program comprehension.

The *mixed model* was supported by the findings of von Mayerhausen, Vans and Somlo (1999), who studied professional programmers in authentic software development projects. They proposed an integrated code comprehension model that combines the results from previous research. The top-down strategy is commonly used, but when the program or the domain is unfamiliar, the comprehension is carried out in a bottom-up manner to gather the program and domain knowledge. With these strategies and using their knowledge base, the programmers build and relate the domain, program and situational models to each other. The proper combination of these three models determines how well the programmer understands the program.

The methodology for studying and analyzing the program comprehension process relies mostly on three approaches: comprehension questions, comprehension summaries and think-aloud protocols.

Comprehension questions are used to determine how well the user has comprehended the program and can remember some aspects of it (Pennington, 1987). These can be used as a measure of comprehension performance. Although program summaries can also be evaluated for analysis of the performance, they are commonly used to investigate the mental models of the program acquired by the programmers during the comprehension task. Previously, *comprehension summaries* have been analyzed from two different standpoints (Pennington, 1987; Good & Brna, 2004). One focus is on the *information types* that are used in the summaries, wherefore ten categories were discovered for different information given from the program. The categories describe several dimensions of the comprehension outcome. First of all, categories can be divided roughly into four knowledge types, namely *functional*, *state*, *data flow* and *control flow* knowledge. Furthermore, functional and state knowledge have different abstraction levels. Functional knowledge has three abstraction levels: *function*, for what the program is used, *action*, function of a part of the program, and *operation*, a statement level description. State knowledge has two levels *state-high* and *state-low* depending on how high is the abstraction level in the description. In addition to the knowledge types, the information types contain four categories for *elaboration*, *meta*-cognition, *unclear* and *incomplete* statements and Byckling et al. (2004) proposed also an *irrelevant* category. The other focus of the summary analysis scheme is on *object descriptions*. They are classified according to their level of description in the summary. For instance, the descriptions are classified in different categories depending on if the objects are described in program or domain specific terms.

The thought sequences and cognitive processing during problem solving are often analyzed with *think-aloud protocol analysis* (Ericsson & Simon, 1984). Resting on an assumption that the sequence of thoughts is not altered during verbalization, protocol analysis has been used successfully in several domains as well as in analysis of the program comprehension processes.

Program Visualization

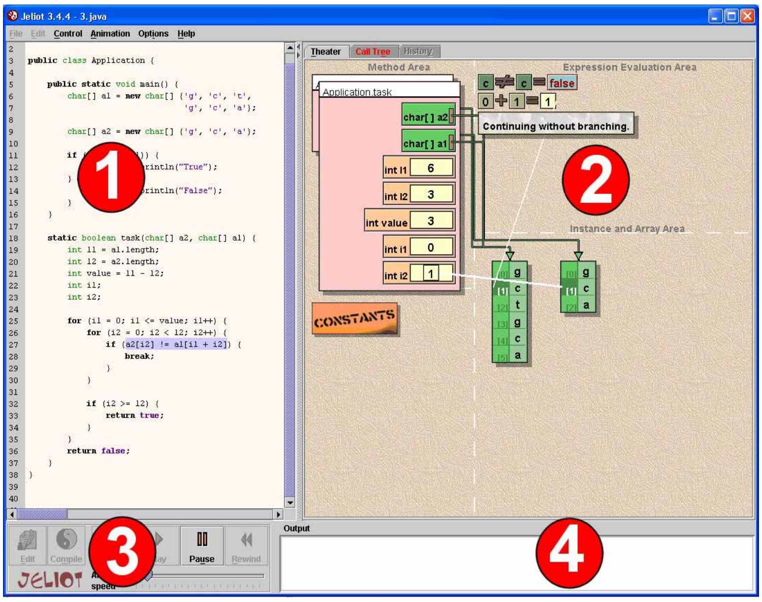
Many systems have been developed in the field of Software Visualization (SV) in the last two decades. Algorithm Visualization (AV), a subset of SV systems, has received the most attention and has been studied relatively intensively. However, the results have been inconclusive (Hundhausen et al., 2002). Program Visualization (PV) is another subfield of SV where visualization is closely coupled with the program and some aspects of the programs execution are visualized either during run-time or post-mortem. These systems are used, for example, to analyze the performance of the software, to debug programs visually or to teach programming concepts. Here we will briefly review systems that are similar in some aspect to Jeliot, the tool used in this experiment.

Javavis (Oechsle & Schmitt, 2002), Jive (Gestwicki & Jayaraman, 2002) and DDD (Zeller & Lütkehaus, 1996) are tools that automatically visualize the programs data flow and part of the control flow during program execution. Javavis and Jive are educational tools whereas DDD is a visual front end to a debugger. To visualize Java programs, Javavis uses UML-like diagrams in their visualizations. It illustrates the run-time behavior of the program through animated object and sequence diagrams. Jive uses a modified contour diagrams and shows the different contexts (i.e. static or dynamic) in which the program is executed together with the source code and local variables. DDD uses its own graph-like format to lay out the data structures and references between them on the screen. Jeliot combines this work to show objects and their fields in a UML-like notation of class diagram. References to the objects are treated as other variables to illustrate the reference semantics of Java language.

The visualization in Dynalab (Boroni et al., 1996) and PlanAni (Sajaniemi & Kuittinen, 2003) concentrates on variables. Dynalab shows the values of variables in a textual format whereas PlanAni shows the variables as graphical objects. Moreover, PlanAni visualizes the variables and operations on them differently, depending on the role (Sajaniemi & Kuittinen, 2003) they are assigned. Dynalab visualizes the programs automatically and can also animate the programs backwards whereas the animations in PlanAni need to be written manually beforehand. Both of these programs are currently developed to visualize only procedural programs. Dynalab has full support for Pascal and restricted support for C and Ada. PlanAni does not restrict the visualized language as long as the animation scripts are described in Tcl/Tk. Jeliot visualizes variables in different scopes separately. For example, method frames contain the local variables of the method, objects contain the fields and static variables are separated in their classes. Jeliot also does not make any difference between the roles of variables. However, it can automatically visualize object-oriented programs.

Jeliot 3

The Jeliot family is a collection of program visualization systems that have been developed over the last ten years (Ben-Ari et al., 2002). The latest version, Jeliot 3, which has been developed at University of Joensuu (Moreno et al., 2004), is designed to help teach novices the programming concepts and to aid in program comprehension and debugging. Its predecessor, Jeliot 2000, has been successfully used to improve the teaching of introductory programming courses, helping novices to acquire vocabulary to explain programming structures and concepts (Ben-Bassat Levy et al., 2003). This might be due to the fact that Jeliot can help the learner to build a viable mental model of the computer executing the program and use it to verbalize the execution.


Jeliot 3.4.4 - 3.java
File Edit Control Animation Options Help
Theater Call Tree History
Method Area
Expression Evaluation Area
Application.task
Continuing without branching.
Instance and Array Area
CONSTANTS
Edit Compile Pause Rewind
JELIOT
speed
Output
1
2
3
4

```

public class Application {

    public static void main() {
        char[] a1 = new char[] {'g', 'c', 't',
                                'g', 'c', 'a'};

        char[] a2 = new char[] {'g', 'c', 'a'};

        if (          )) {
                      rintln("True");
        }
                      rintln("False");
        }
    }

    static boolean task(char[] a2, char[] a1) {
        int l1 = a1.length;
        int l2 = a2.length;
        int value = l1 - l2;
        int i1;
        int i2;

        for (i1 = 0; i1 <= value; i1++) {
            for (i2 = 0; i2 < l2; i2++) {
                if (a2[i2] != a1[i1 + i2]) {
                    break;
                }
            }

            if (i2 >= l2) {
                return true;
            }
        }
        return false;
    }
}
```

FIGURE 1
User interface of Jeliot 3. (1 = *code editor*; 2 = *visualization view*; 3 = *control panel*; 4 = *output console*.)

Jeliot 3 retains the novice-oriented GUI and animation display of Jeliot 2000 by only adding new menus to expose more functionality to the user, especially when it is used during lectures. In order to make the system extensible and to allow adding new features into the visualization, Jeliot 3 introduced a new design. It automatically visualizes the execution of user-written Java programs by illustrating the data and control flow and object-oriented features of the program. Jeliot 3 can visualize a large subset of novice-level Java programs and it is freely distributed under GPL (see http://cs.joensuu.fi/jeliot/). The user interface of Jeliot 3 is shown in *Figure 1* (1 = *code editor*; 2 = *visualization view*; 3 = *control panel*; 4 = *output console*).

The interface consists of four different areas. A *code editor* on the left side shows the program code, and during program visualization, the currently executed statement or expression is highlighted. A *control panel* in the bottom left corner is used to control the animation with VCR-like buttons. On the right side of the window, the largest area of the Jeliot's window is occupied by a *visualization view* showing the execution state of the program. Moreover, the animation view is further divided into four different areas:

- *the method area* showing the currently executed method and local variables;
- *the expression evaluation area* where the expressions are evaluated step-by-step and messages are shown to the user;
- *the constant area* containing classes together with their static variables and the constant box from which the literal constants appear; and,
- *the instance and array area* showing the visualization of the arrays and instances containing their fields.

Finally, an *output console* is located in the bottom right corner of the window, showing the output of the executed program. Furthermore, there are two specialized visualizations, where only the call tree or the execution history of the program is shown, on separate tabs of the tabbed pane. These views reduce the complexity and the amount of information shown to the users and thus help them to concentrate on the relevant parts. In the call-tree visualization, the previous method calls are shown and the currently active methods are highlighted. This allows following the program execution in the level of method calls and further reduces the complexity. In the history view, users can analyze the previous stages of the execution in a step-by-step manner and thus are able to reason about how the current execution step was reached.

In a typical session with Jeliot, the user either writes or loads a program. The user can then compile the program by using the user interface of Jeliot. When the program is compiled, a visualization view, where the user can see the animation of the program execution, is opened. Jeliot shows the execution either step by step or continuously. The user can control the animation with the buttons, for instance by stopping the animation and continuing in a step-wise manner. Reverse execution is not possible; but the user can view the execution history which is a stepwise recording of the current execution.

Eye-movement tracking

We move our eyes in order to bring an image of the inspected object onto the fovea, a small and high-resolution area of the retina. Once the image of the object is stabilized on the retina, the information can be extracted. This way the visual attention is linked with the current direction of eye gaze and most of the time it is also diverted to the point of visual inspection. Following this eye-mind assumption, if we can track the movements of the eyes, we can also obtain good insights into and investigate the path and focus of visual attention during a task. Previous research has firmly established this relation between eye movements, visual attention and underlying cognitive processes (Just & Carpenter, 1976; 1980; Rayner, 1998). Knowing which objects have been visually inspected and in which order and context, we can attempt to infer what cognitive processes were involved in performing a task related to these objects.

An eye tracker is a device that records eye movements. To estimate the direction of gaze, most of the current eye trackers use infrared light emitters and video image analysis of the center of the pupil and reflections from the cornea. Typically, the accuracy of the eye trackers currently available commercially is around 1 degree, while the data are sampled at rates of 50–500Hz. Modern eye-trackers are relatively cheap and able to collect gaze data reliably and unobtrusively. Two general classes of eye tracking devices exist: a remote optics, table-mounted version and a head–mounted optics with a see-through mirror. Regardless of the option, both types of eye trackers must be calibrated for each user before the first experiment.

From the signal obtained from an eye tracker, the two most important types of eye movements usually identified are saccades and fixations (Salvucci & Goldberg, 2000; Sibert & Jacob, 2000). Saccades are rapid ballistic movements of eyes that are executed to reposition the eyes from one location of attention to another. A single saccade can last between 30 and 120 ms and can span over 1 to 40 degrees of visual angle (Sibert & Jacob, 2000), with velocities ranging up to 500 degrees per second (Rayner, 1998). No visual information is extracted during a saccade; this is called saccadic suppression (Matin, 1974). Fixations are the movements of eyes stabilizing the image of an object on the retina, providing the human visual system a possibility to extract the features of the object. Typically, the fixation duration ranges from 200 ms to 300 ms (Rayner, 1998). It is assumed that during the period of a single fixation the information is extracted, decoded, and interpreted. The fixation duration can therefore be thought to be related to the processing required to extract and interpret the information (Just & Carpenter, 1976; Goldberg & Kotval, 1998; 1999).

Accurate measurement and analysis of eye movements in terms of saccades and fixations therefore provide us with the details of cognitive processing and related allocation of visual attention within a performed task.

For instance, the fixation count or the sum of fixation durations on a certain element can be related to the importance of that element. The fixation duration might also be seen as a measure of cognitive workload. As Goldberg and Kotval (1999) reported, the fixation duration increased when the task required difficult cognitive processing. In the context of program visualization interfaces, the measure of relative fixation count can, for instance, correspond with the relative importance of a representation (e.g. a code or a state diagram) of a program to the current user of the tool.

Previous research of eye movement tracking

In many fields of HCI and in other domains, studies of eye-movement tracking have significantly contributed to the body of available knowledge. Eye-movement tracking has been successfully employed in studies of reading (Just & Carpenter, 1980, 1984; Rayner, 1994), gaze-based interaction (Jacob, 1993; Karn & Jacob, 2004), eye typing (Majaranta & Räihä, 2002), menu selection (Crosby & Peterson, 1991; Aaltonen, Hyrskykari & Räihä, 1998), usability (Goldberg & Kotval, 1998; 1999) or in virtual reality (Duchowski et al., 2000, 2002).

Possible differences in the eye movement patterns of experts and novices have been of great interest. Previous studies of eye movement patterns in domains other than program visualization have shown that (a) search strategies differ between novice and expert radiologists (Nodine and Mello-Thons, 2000), and (b) the eye movement patterns of expert-pilots were better defined and the dwell times were significantly shorter than those of novices (Kasarskis et al., 2001). The common conclusion of these and other reports is that the domain knowledge and experience of participants seem to be the main factors influencing both the performance and the related gaze behavior.

In the context of the psychology of programming, however, only a few attempts to utilize eye-movement tracking have been made. Crosby and Stelovsky (1989, 1990) studied aspects of visual attention during reading the programs and program comprehension. In these studies, an eye tracker was used to discover the relationship between cognitive styles and individual differences, and code-reading patterns. In Crosby and Stelovsky's studies, novices and experts were eye tracked during algorithm comprehension. With the help of eye tracking metrics, complex statements in the source code of a variation of binary search algorithm have been found and related to beacons (Crosby et al., 2002). Moreover, differences in the programming experience of participants were reflected in the times they spent viewing different areas of particular complex statements. However, only one representation of a program was used (the code), and the focus of the research was mainly on the critical, but surface features of the code, not on the behavior during dynamic visualization of the program.

Program visualization often involves simultaneous presentation of several different representations of a program in adjacent views. Previous studies, investigating issues such as visual attention switching or a representation use during program comprehension aided by visualization, involved only static pre-computed stimuli and the analysis was based on a recording of mouse movements over a blurred interface (Romero et al, 2002; 2003). In comparison to the eye movement tracking, the validity of such an approach was shown to be questionable (Bednarik & Tukiainen, 2004). Although there is an apparent need for a deeper investigation of gaze behavior and the underlying cognitive processes during program visualization (Chandler, 2004), no attempts were made to fill this gap.

EXPERIMENT

The present research investigates differences in gaze behavior of participants with different levels of programming experience during program animation. In a preliminary study (Bednarik et al., 2005) we discovered that, in terms of global attention allocation between code and visualization, novice and expert gaze patterns do not differ during comprehension aided by program animation. This result was quite surprising, since in other domains experts' and novices' gaze patterns differ significantly. However, we found some differences in the mean fixation durations. To further investigate these findings, we deconstructed the interface of the visualization tool and compared the performance of novice and expert participants in terms of their gaze behavior. This provides us with specific knowledge of how explicit areas are used by different programmers and whether their use interacts with performance and cognitive models.

To examine this hypotheses, we conducted an empirical experiment using a remote eye tracker to record the gaze behavior of participants during a program-comprehension task aided by an animation. Two groups of participants with different levels of experience used Jeliot to comprehend three short Java programs while their eye movements were simultaneously tracked.

METHOD

A mixed one-between-subject (experience) with two levels and one-within-subject (program) with two levels design was used. The dependent variables were relative fixation count over the areas of interest, number of switches per minute, mean fixation durations over the areas of interest and overall. The fixation count is a measure related to the level of the participant's interest in an area and therefore related to the importance of that area. The number of switches per minute is a measure of attention allocation dynamics. The mean fixation duration is associated with the depth of processing required to understand an attended element. The analysis of eye movement protocols used only data recorded during the program animation because that was the only time when all the representations were shown concurrently and selection of the attended-to representation could make a difference in understanding the program.

The process of program comprehension was recorded with a video camera and the interaction protocol with Jeliot was logged. Program summaries were analyzed using the program summary analysis scheme of Good and Brna (2004) and revised by Byckling et al. (2004). The summaries were scored from 0-3 according to the completeness and correctness criteria. Most of the statistical analysis was carried out using repeated measures ANOVA and planned comparisons based on t-test.

Participants

Eighteen participants were recruited from high-school students attending a university level programming course, undergraduate and graduate computer science students from the local university; each received a lunch ticket. Due to technical problems, data from two participants was discarded. The results are based on data collected from 16 subjects (13 male, 3 female).

Participants were divided into two groups according to their level of programming experience. Those with less than 24 months of programming experience were regarded as novices and those with more than 24 months as experts. The characteristics of the two groups are presented in *Table 1*. Groups' mean values for programming experience (in months) and Java experience (in months) and counts for previous experience with Jeliot (yes=1, no=0) and previous experience as a professional programmer (yes=1, no=0) are shown. Standard deviations are in parentheses.

TABLE 1
Characteristics of the groups. * marks significant difference between groups in two-tailed t-test (interval values) or χ^2-test (nominal values) with $p<0.05$.

Experience level	Count	Prog. exp.*	Java exp.*	Jeliot exp.	Prof. exp.
novices	8	12.75 months (6.90)	6.38 months (4.60)	3	1
intermediates/ experts	8	85.50 months (56.44)	19.75 months (15.00)	2	1

Materials and apparatus

Three short Java programs - factorial computation (program 1), recursive binary search program (program 2) and naïve string matching (program 3) - were presented to participants. The lengths of the programs (lines of code) were 15, 34, and 38, respectively. Each of the programs generated one line of output and did not require user input. The names of the methods and variables were altered so that recognition of a program based on surface features was difficult. To allow for comparison with the eye tracking data, we used an adapted version of Jeliot 3 that logged all the user actions and all the changes in the visualization of the programs in this study.

The remote Tobii ET-1750 (50Hz) eye tracker, which made no contact with participants, was used to track eye movements; the eye tracker is built into the TFT panel so no moving part is visible and no sound can be heard during recording. The interaction protocol (such as key-strokes and mouse clicks) was collected for all target programs, and audio and video were recorded for a whole session. The minimal duration of fixation for the algorithm processing eye-data was set at 100ms. Seven static areas of interest (AOI), matching the seven main areas in the Jeliot interface, were defined: the code, the expression evaluation area, the method area, the instances area, the constants, the control, and the output area.

Procedure and design

The experiment was conducted in a quiet usability laboratory. Participants were seated in an ordinary office chair near the experimenter and facing a 17" TFT display. Every participant then passed an automatic eye-tracking calibration. The calibration required the participants to follow sixteen shrinking points that appeared one by one across the screen. If needed, the calibration was repeated in order to achieve the highest possible accuracy. The settings of the experiment are shown in *Figure 2*.

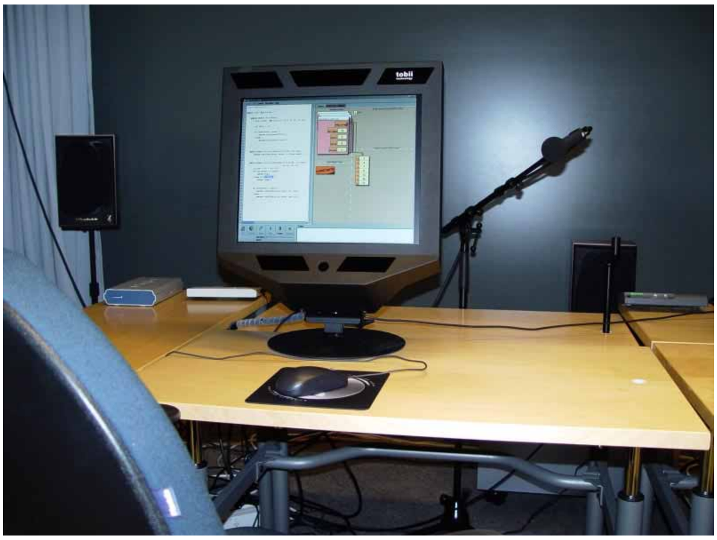


FIGURE 2
Experimental settings in the laboratory.

After successful calibration, participants performed three sessions, each consisting of a comprehension phase using Jeliot 3 and a program summary writing phase. Participants were instructed to comprehend the program as well as possible, and they could interact with Jeliot as they found necessary. The target programs contained no errors and were always preloaded into Jeliot and compiled. The duration of a session was not limited.

The first program was a factorial computation that was used as a warm-up; the resulting data were discarded. The order of the two comprehension tasks (program 2 and program 3) was randomized so that half of the participants started with the recursive binary search and the other half with naïve string matching.

RESULTS

Completion and animation times

Mean completion times for the comprehension phase were 17.6 minutes (SD = 10.0) for novices and 9.8 minutes (SD = 2.6) for experts; according to a two-tailed t-test, the differences were statistically significant (t(14) = 2.23, p<*.05*). From that time, novices spent, on average, 85.4% (SD = 9.6) of their time animating the program whereas experts spent 52.9% (SD = 20.0) of their time animating the program; according to a two-tailed t-test, the difference was statistically significant (t(14) = 4.13, p<*.01*).

Interaction patterns

The mean number of clicks on each of the buttons on the control panel and the mean number of animation replays between groups are show in *Table 2*. Novices interacted with the user interface more than experts did. Novices played, rewound and replayed the animation significantly more.

TABLE 2
Mean values of interaction with the user interface (standard deviations in parentheses).

Button	**Pause**	**Play**	**Rewind**	**Step**	**Overall**	**# of replays**
Novices	3.22 (3.55)	5.28 (3.69)	1.83 (1.09)	39.56 (57.20)	49.89 (63.42)	2.722 (0.939)
Experts	1.25 (1.00)	2.19 (1.19)	0.63 (0.69)	2.50 (7.07)	6.57 (7.55)	1.625 (0.694)
t-value (df=14)	1.395 *ns*	2.144 p=*.05*	2.824 p=*.01*	1.761 p=*.10*	1.91 p=*.09*	2.71 p=*.01*

The interaction patterns of a typical comprehension session of program 3 for a novice and expert (*Figure 3*) show the states of the visualization tool when used by a typical novice and by a typical experienced programmer. In both of the comprehension tasks, the experts spent significantly more time on initial code reading (on average, 173 seconds) before they animated a program for a shorter time (on average, 340 seconds) and usually at high speed. On the other hand, the novices paid little attention to the code (on average, 45 seconds) and it took them significantly more time to view the animation than it took the experts (on average, 857 seconds). In addition, behavior of a novice in our study was characterized by frequent use of a pausing/stepping approach combined with more replays of the animation. From the protocols and gaze-recordings it is clear that the stepping and pausing occurred around the central parts of the currently comprehended algorithm.

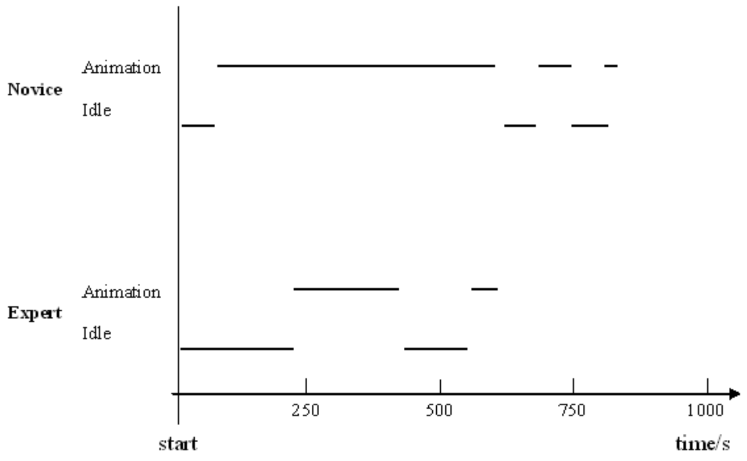


FIGURE 3
The phases of Jeliot animation engine during the comprehension of the binary search program.

Analysis of comprehension summaries

The comprehension summaries were graded in a scale from 0 to 3 and the averages of the two groups are shown in *Table 3*. The results indicate that experts received more points than novices did; however, the differences were not statistically significant.

TABLE 3
Points received from the summary evaluation (standard deviations in parentheses).

	program 2	**program 3**
both groups	1.81 (0.75)	1.88 (0.96)
novices	1.63 (0.52)	1.50 (1.07)
experts	2.00 (0.93)	2.25 (0.71)

In order to understand the qualitative differences in the summaries better, they were analyzed according to the scheme described by Good and Brna (2004). We analyzed only the information types found in the summaries (*Table 4*). The first eleven rows describe the standard categories of the analysis scheme. The repeated measures analysis of variance revealed no significant effect of program (F(1,14) = 2.32, *ns*) or experience (F(1,14) = 2.32, *ns*) on the information types contained in the summaries. The interaction between program and experience was not significant (F(1,14) = 2.32, *ns*). However, an effect of the information type was discovered (F(9, 126) = 6.82, p<*.001*), but had no interaction with the level of experience (F(9, 126) = 0.71, *ns*). The interaction effect between program and information type was significant (F(9, 126) = 2.45, p=*.013*).

To investigate this difference, pairwise comparisons were run. According to these comparisons, the action, and the irrelevant categories were present significantly more in the summaries of program 3 (t(14) = 2.38, p<.05) and t(14) = 2.45, p<.05, respectively), while the control related statements were present more in the summaries of program 2 (t(14) = 2.36, p<.05). No other significant difference was found, however, the state-low statements were more often present in program 2 summaries, with probability approaching a significance level of 0.05, (t(14) = 1.99, p=*.066*).

TABLE 4
Information type analysis (standard deviations in parentheses; unused categories omitted).

	program 2		**program 3**	
	novices	**experts**	**novices**	**experts**
function	5.68 % (5.71)	5.96 % (4.59)	5.23 % (4.86)	6.17 % (4.84)
action	6.71 % (6.27)	5.68 % (6.78)	13.61 % (13.71)	9.87 % (9.49)
operation	17.75 % (19.73)	10.63 % (17.17)	15.69 % (13.09)	16.28 % (14.37)
state-high	5.70 % (6.36)	8.13 % (7.47)	6.73 % (3.09)	15.91 % (10.10)
state-low	12.59 % (15.36)	12.83 % (11.80)	11.99 % (13.07)	6.78 % (5.90)
data	8.84 % (7.43)	22.90 % (8.11)	10.47 % (8.36)	9,91 % (8,67)
control	23.65 % (21.73)	21.00 % (5.11)	10.57 % (6.57)	12,33 % (10.25)
elaborate	16.45 % (12.41)	10.78 % (12.20)	16.22 % (13.13)	19,63 % (15.47)
meta	0.96 % (2.72)	0.83 % (2.36)	5.21 % (9.76)	0,00 % (0,00)
irrelevant	0.00 % (0.00)	1.25 % (2.48)	4.28 % (5.50)	3.11 % (4.58)
info-high	20.22 % (12.70)	36.99 % (13.50)	22.42 % (8.72)	32.00 % (12.94)
info-low	53.99 % (25.53)	44.46 % (16.46)	38.25 % (17.88)	35.38 % (22.14)

The last two rows of *Table 4* contain two aggregate values that describe the level of abstraction used in the summaries. The value *info-high* is a measure of high abstraction and is a sum of function, state-high and data statements. On the other hand, *info-low* is a sum of operation, state-low and control and indicates a low level of abstraction in the summary. The abstraction levels were used differently (F(1,14) = 5.37, p<*.05*). Experts used more *info-high* statements and less *info-low* statements, but there was no interaction between abstraction level and experience (F(1,14) = 2.21, *ns*).

Eye-gaze related data

Since the programs did not generate an extensive output, some of the participants were not gazing at the output area. Therefore, only the data of those participants who produced at least one fixation to each of the areas were included in the analysis. As one experimental participant did not perform any switching during comprehending the program 2, data for only fifteen participants were used in this analysis of eye-gaze data.

Distribution of Fixations

The distribution of the fixations over different areas of interest (AOI) during animation is illustrated in *Figure 4*, the value of the visualization AOI is composed as a sum of the method, expression, instance, and constants areas of interest. Only the data of those participants who produced at least one fixation to each of the areas were included in the analysis.

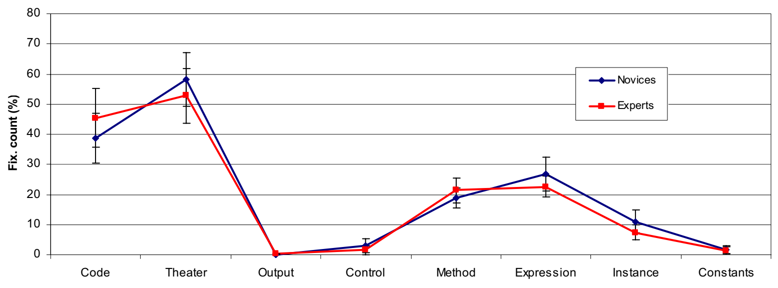


FIGURE 4
Relative fixation-count distribution during animation. The value of Visualization column represents a sum of fixations on the method, expression, instance, and constants areas.

According to the repeated measures ANOVA, the use of areas of interest was significantly different (F(5,65) = 62.89, p<*.001*). The effect of program was not significant (F(1,13) = 0.001, *ns*) and the effect of experience to the fixation count distribution was also not significant (F(1,13) = 0.46, *ns*). However, the interaction between program and AOI was significant (F(5,65) = 9.00, p<*.001*). Other interactions were not significant.

Distribution of the fixations over the user interface during comprehension of program 3 is shown in *Figure 5* as hotspot visualization, where the areas with more fixations are in darker colors. The distributions were similar for both groups. During both programs there were two areas that got the highest attention from both groups. One of the areas was in the code where the central idea of the program lay, and the other was in the expression area of the visualization frame.

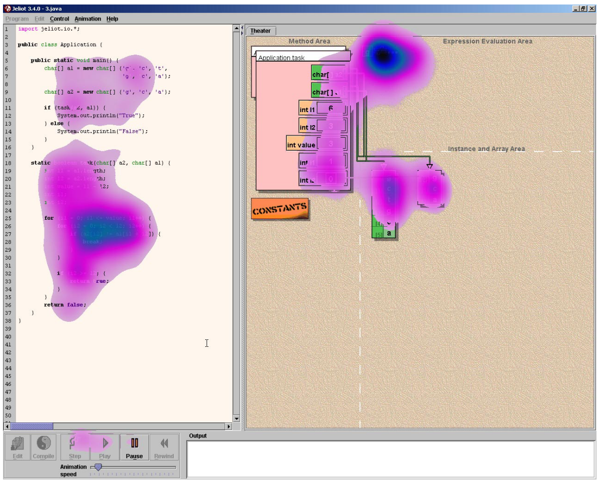


FIGURE 5
Visualization of the fixation distribution for the string matching program; the more attended areas are shown in darker colors.

Switching behavior

Figure 6 shows the mean number of switches per minute during animation. A switch was measured every time a gaze location changed between any two of the seven areas of interest. As one experimental participant did not perform any switch with the program 2, there were only data for fifteen participants used in this analysis. For the data during the animation, a 2 x 2 x 21 (experience x program x switch type) ANOVA showed a significant main effect of switch type (F(20, 260) = 68.44, p<*.001*), and no effect of program (F(1,13) = 1.04, *ns*) on the switching frequency. The effect of experience was not significant, however, approaching near significance (F(1, 13) = 3.37, p = *.089*). The interactions between program and experience, and between switch type and experience were not significant (F(1, 13) = 0.50, *ns*), and (F(20, 260) = 1.02, *ns,* respectively). However, the interaction between program and switch type was significant (F(20,260) = 11.38, p<*.001*).

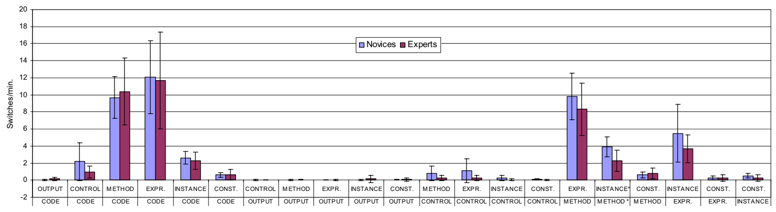


FIGURE 6
Visual attention switching during animation; * indicates significant difference in the two-tailed t-test (p<.05).

The pairwise comparisons revealed the sources of the differences: the switches between the code and the method AOIs and the code and the instances AOIs differed significantly between the programs (t(13) = 3.07, p<*.01*), and (t(13) = 3.62, p<*.005*, respectively). Some switches within the animation frame, particularly those between the expression and the method areas, between the instances and method areas, and between the expression and instances areas, differed significantly (t(13) = 2.24, p<*.05*), (t(13) = 2.63, p<*.03*), and (t(13) = 6.97, p<*.001*, respectively).

Fixation durations

Figure 7 shows the mean fixation durations during animation for the seven main areas of interest (AOI) and the overall mean fixation duration.

These have been computed as the sum of durations of all fixations landing at an area of interest divided by the number of fixations. Since the programs did not generate an extensive output, some of the participants were not gazing at the output. The data on the areas affected during the animation, the method, the expression evaluation, the instance, and the constants areas, were included into the analysis.

According to repeated measures ANOVA, the durations of fixations on different AOIs were significantly different (F(4,52) = 16.66, p<*.001*). On almost all areas, experts seemed to fixate significantly shorter (F(1,13) = 7.27, p=*.018*) and an interaction effect between experience and fixation duration on the areas was significant (F(4,52) = 3.88, p<*.01*). No effect of program on the mean fixation duration was found (F(1,13) = 0.495, *ns*), as well as no interaction between program and AOI (F(4,52) = 1.59, *ns*).

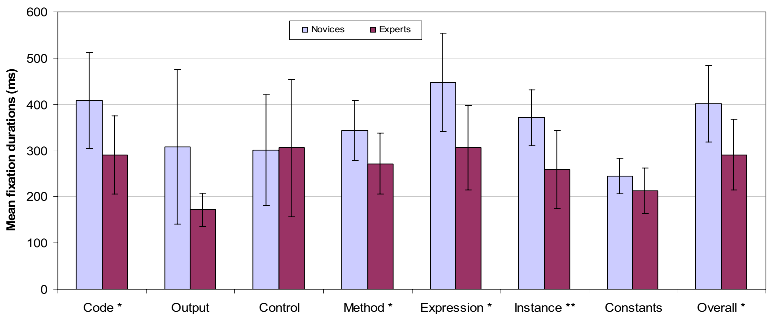


FIGURE 7
Fixation duration during animation. * indicates significant difference between groups in the two-tailed t-test (p<.05) and ** the two-tailed t-test (p<.01).

DISCUSSION

In our study, expert programmers completed the comprehension phase faster than novices did. This can be kept as one measure of performance. From the analysis of the summaries it can be seen that, although the differences were not statistically significant, experts performed better than novices. In the qualitative analysis of the summaries, it was found that, compared to novices, experts used higher level of abstraction in their summaries, but also this difference was not statistically significant. However, according to previous studies, the use of high level references is a sign of expertise (Hoadley et al., 1996; Pennington, 1987). Together with the finding that experts comprehended the programs faster, these indicate better performance of the experts compared to the novices.

From the interaction protocols analysis we found that the experts usually started their comprehension task by reading the code and spending significantly larger proportion of the session on constructing hypotheses. This finding is in agreement with results obtained in previous studies (Romero, 2003; 2002). After these initial code-reading episodes, the experts spent significantly less time than novices on animating the programs. Novices interacted with the user interface more than the experts did. This is due to the fact that novices spent more time on visualizing the program than experts did, and novices also replayed the animation more, compared to experts. The number of replays indicates that novices relied on visualization whereas experts used it as an additional source of information. Experts spent less time animating the programs which was in agreement with the hypothesis that experts would concentrate more on code reading. This happened only before and after they had been visualizing the program. The code-reading episodes could have affected the gaze behavior of the experts during the program animation compared to novices. However, we have not found such projection into the fixation patterns in terms of fixation distribution during the animation.

The results of this experiment related to gaze behavior during animation show that the level of experience does not affect the distribution of fixation counts, while the switching behavior is affected slightly. This result is inconsistent with previous eye movement studies where the differences between novice and expert gaze behavior were investigated. The animation attracts novice and expert programmers almost equally to attend it in quite similar patterns. Most of the animation time was spent on viewing the visualization part of the Jeliot's user interface. For more experienced participants, the distribution of fixations between code and animation was slightly more balanced, in other words, the novice programmers relied more on visualization.

During animation, the attention switch between the code and the expression evaluation areas was most common, followed by the switch between the code and the method area. The switches to and from the control area were higher for novices. We connect this difference to the fact that during the animation novices were interacting with the tool more than experts were. According to the analysis, the effect of experience on the switching behavior was only a nearly significant.

Unlike the fixation counts and attention switching, we found a significant effect of experience on the fixation duration. For all the main areas and also overall, the fixation duration of experts was shorter than that of novices. The fixation duration when participants gazed at the control buttons was the same regardless of experience. Altogether this means that novices needed to devote significantly more time to comprehending a currently animated feature, which is naturally linked to expertise. This difference was greatest when participants were attending to the instance area and smallest for the area where constants were appearing.

Considering the gaze behavior as related to a target program, we found that the use of the discrete areas and therefore the type of the attention switch differed as the comprehended program changed. In previous studies on the coordination of multiple representations during a debugging task (Romero et al., 2003; 2003) it has been suggested that the balanced use of different representations of a program during comprehension might be linked to superior programming experience. As seen from the fixation count distribution and switching behavior, both groups used the provided representations in about the same balanced way. Therefore, it could be, provocatively, suggested that using the Jeliot also made the novice programmers to behave in patterns similar to those of experts. This view could be supported by the difference found in the mean fixation durations: although the animation was attended in similar patterns, novices spent significantly more time to process the animation. However, the performance, as measured by the comprehension summaries, was different but not significantly.

FUTURE WORK

The results obtained in this initial experiment need to be confirmed further and extended. Future research can take several directions. Our aim is to investigate the effects that a single animation element and the difficult sections of code have on gaze behavior as well as the changes in gaze behavior over time and with increasing experience. The ultimate goal is to support learning with dynamic program visualizations that could adapt to the needs of the users. We believe that using the eye-tracker to collect the actual gaze directions and use it to adapt visualization is a way to get closer to our goal.

In addition, we plan to develop better methodological grounds for applying the eye-tracking in the context of program visualization. As seen from the present study, some of the widely used eye-tracking metrics are not sufficiently able to discover the links between experience, produced mental models and gaze behavior. Therefore, available measures need to be related to the cognitive processes and outcomes and new measures have to be developed. For example, we plan to examine, whether and how attending a certain element or area of animation correlates with the produced mental models captured in program summaries. Another direction could be taken, for instance, when eye tracking is applied to program visualization in real time: the level of dissociation between the attention focus of a programmer and the currently animated location should be investigated. These studies will provide a deeper understanding of the cognitive processes involved in program comprehension during program visualization and will further stimulate the research on the adaptivity of visualization tools.

Adapting program visualization

When visualizing a program, users of the visualization tools have different needs and levels of knowledge. Currently, a user can change some aspects of the visualization manually, but there are only few systems that would automatically adapt to the user's previous experience or needs. This would be beneficial for the user because her cognitive load could be reduced by hiding details that are not relevant to the current task of users or are already known to the users.

There are several scenarios how Jeliot could be made more adaptive. First of all, we could collect data from a learning environment for programming and use user modeling to analyze which programming concept the user currently does or does not understand. With the information on user knowledge, we could change the granularity of the visualization to emphasize those concepts that the users are currently struggling with and minimize the details of those that they already understand (Brusilovsky & Su, 2002).

The visualization could be adaptive in real time, based on the data collected from the user, for example, via an eye tracker. With an eye tracker, one can track the gaze path during a comprehension task and therefore can obtain insights into the user's allocation of visual attention and attention switching between different areas in an interface. Knowledge of gaze-related patterns provides us with important aspects of the underlying cognitive processes and could be a hint as to the needs of the users. As it has also been argued elsewhere (Cross et al., 1999), supporting those cognitive processes that are beneficial to program visualization, and therefore also to comprehension, can yield better effects on learning. Thus, one of the aims of the research presented in this paper was to discover the basic patterns of eye movements during program comprehension supported by dynamic visualization and thereby provide the research community with a starting point and means of adapting and tailoring the visualizations to personal needs. Till now, the eye- movement research has mainly concentrated on utilization of voluntary eye movements as an alternative gaze-based selection and pointing technique for HCI, but little has been done to exploit the gaze direction during complex dynamic visualizations.

CONCLUSION

We conducted an empirical experiment to discover the similarities and differences between experienced and novice programmers in interaction, in the comprehension process, and in the gaze behavior during program comprehension aided by a visualization tool. We employed non-intrusive remote eye tracking equipment to record the eye movements of programmers.

Our results in terms of attention switches between representations and distribution of fixations show no significant difference in behavior between novice and expert groups of programmers during program animation. The focus of visual attention seems to be distributed in space and also most of the time evenly, regardless of previous experience with programming. When the level of processing required to attend to the animation is measured as the duration of fixations over the main areas of interest and overall, our results show a significant difference.

Acknowledgements

We thank all participants for taking part in this study. We acknowledge Andrés Moreno for help with preparation of this study. We thank the reviewers for comments to improve the manuscript.

REFERENCES

Aaltonen A., Hyrskykari A., Räihä, K.-J. (1998). 101 Spots, or How Do Users Read Menus? In *Proceedings of the CHI'98* (pp.132–139), NY, ACM.

Basili, V. R., Mills, H. D. (1982). Understanding and Documenting Programs. *IEEE Transactions on Software Engineering*, 8(3), (pp. 270–283).

Bednarik, R., Tukiainen, M. (2004). Visual attention tracking during program debugging. In *Proceedings of NordiCHI'04* (pp. 331–334), NY: ACM.

Bednarik, R., Myller, N., Sutinen, E., Tukiainen, M. (2005) Effects of Experience on Gaze Behaviour during Program Animation. *Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop* (PPIG'05) (pp. 49-61), Brighton, UK, June 28 - July 1, 2005.

Ben-Ari, M., Myller, N., Sutinen, E., Tarhio, J. (2002). Perspectives on Program Animation with Jeliot. In S. Diehl (Ed.), *Software Visualization* (pp. 31–45), Berlin: Springer-Verlag.

Ben-Bassat Levy, R., Ben-Ari, M., Uronen, P.A. (2003). The Jeliot 2000 Program Animation System. *Computers & Education*, 40(1), (pp. 1–15).

Boroni, C. M., Eneboe, T.J., Goosey, F.W., Ross J. A., Ross, R. J. (1996) Dancing with Dynalab - Endearing the Science of Computing to Students. In *Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education* (pp. 135–139), NY: ACM.

Brusilovsky, P. and Su, H.-D. (2002). Adaptive Visualization Component of a Distributed Web-based Adaptive Educational System. In *Proceedings of 6th International Conference on Intelligent Tutoring Systems 2002* (pp. 229–238), Berlin: Springer-Verlag.

Byckling, P., Kuittinen, M., Nevalainen, S., Sajaniemi, J. (2004). An Inter-Rater Reliability Analysis of Good's Program Summary Analysis Scheme. In *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group* (pp. 170–184).

Crosby, M. and Peterson, W., (1991). Using eye movements to classify search strategies. In *Proceedings of the Human Factors Society 35th Annual Meeting* (pp. 1476–1480).

Crosby, M. E., Scholtz, J., Wiedenbeck, S. (2002). The Roles Beacons Play in Comprehension for Novice and Expert Programmers. In J. Kuljis, L. Baldwin, R. Scoble (Eds.), *Proceedings of the 14th Annual Workshop of the Psychology of Programming Interest Group* (pp. 58–73).

Crosby, M. and Stelovsky, J. (1989). The influence of user experience and presentation medium on strategies of viewing algorithms. In *Proceedings of 22nd Annual Hawaii International Conference on System Sciences* (pp. 438–446).

Crosby, M. and Stelovsky, J. (1990). How do we read algorithms? A case study. *IEEE Computer*, 23(1), (pp. 24–35).

Cross, J. H., Hendrix, T. D., Mathias, K. S., Barowski, L. A. (1999). Software Visualization and Measurement in Software Engineering Education: An Experience Report. In *Frontiers in Education Conference FIE '99* (vol. 2, pp. 12B1/5–12B110).

Detienne, F. (2002). Software Design – Cognitive Aspects. Springer-Verlag London.

Duchowski, A.T., Shivashankaraiah, V., Rawls, T., Gramopadhye, A. K., Melloy, B., Kanki, B. (2000). Binocular eye tracking in virtual reality for inspection training. In *Tracking Research & Applications Symposium 2000* (pp. 89–96).

Duchowski, A. T., Medlin, E., Cournia, N., Gramopadhye, A. K., Melloy, B., Nair, S. (2002). 3D eye movement analysis for VR visual inspection training. In *Tracking Research & Applications Symposium 2002* (pp. 103–110).

Ericsson, K. & Simon, H. (1984). *Protocol Analysis*. Cambridge, MA: MIT.

Gestwicki, P., Jayaraman, B. (2002). Interactive visualization of Java programs. In *Proceedings of IEEE Symposia on Human Centric Computing Languages and Environments* 2002 (pp. 226–235).

Goldberg, J. H., Kotval, X. P. (1998). Eye Movement-Based Evaluation of the Computer Interface. In S. K. Kumar (Ed.), *Advances in Occupational Ergonomics and Safety* (pp. 529–532), Amsterdam: IOS.

Goldberg, J. H., Kotval, X. P. (1999). Computer Interface Evaluation Using Eye Movements: Methods and Constructs. *International Journal of Industrial Ergonomics*, 24(6), (pp. 631–45).

Good, J., Brna, P. (2004). Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. *International Journal of Human-Computer Studies*, 61, (pp. 169–185).

Hoadley, C. H., Linn, M. C., Mann, L. M., Clancy, M. J. (1996). When, Why and How Do Novice Programmers Reuse Code? In W. D. Gray and D. A. Boehm-Davis (Eds.), *Empirical Studies of Programmers: Sixth Workshop* (pp. 109–129), NJ: Ablex.

Hoc, J.-M., Green, T.R.G., Samurcay, R., Gilmore, D.J. (Eds.) (1990). Psychology of programming. Academic Press.

Hundhausen, C. D., Douglas, S. A., Stasko, J. T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing* 13(3), (pp. 259–290).

Jacob, R. J. K. (1993). Eye Movement-Based Human-Computer Interaction Techniques: Toward Non-Command Interfaces. In H. R. Hartson, D. Hix (Eds.), *Advances in Human-Computer Interaction* (Vol. 4, pp. 151–190).

Just, M. A. and Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology*, 8, (pp. 441–480).

Just, M. A. and Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review*, 87(4), (pp. 329–354).

Just, M. A. and Carpenter, P. A. (1984). Using Eye Fixations to Study Reading Comprehension, in D. Kieras & M. Just (eds), *New Methods in Reading Comprehension Research*, Lawrence Earlbaum Associates, Hillsdale, New Jersey, (pp. 151-182).

Kasarskis, P., Stehwien, J., Hickox, J., Aretz, A., and Wickens, C. (2001). Comparison of expert and novice scan behaviors during VFR flight. In *Proceedings of the 11th International Symposium on Aviation Psychology*.

Majaranta, P., Räihä, K-J. (2002). Twenty Years of Eye Typing: Systems and Design Issues. In *Eye Tracking Research & Applications Symposium 2002* (pp. 15–22).

Matin, E. (1974) Saccadic suppression: a review and an analysis. *Psychological Bulletin*, 81(12), (pp. 889–917).

Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M. (2004). Visualizing Programs with Jeliot 3. In *Proceedings of Advanced Visual Interfaces, AVI 2004* (pp. 373–376).

Nodine, C., Mello-Thoms, C. (2000). The nature of expertise in radiology. In J. Beutel, H. Kundel, R. Van Metter (Eds.), *Handbook of Medical Imaging*, WA: SPIE.

Oechsle, R., Schmitt, T. (2002). JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). In S. Diehl (Ed.), *Software Visualization*, Vol. 2269 of LNCS, (pp. 176–190). Berlin: Springer-Verlag.

Pennington, N. (1987). Comprehension Strategies in Programming. In G. M. Olson, S. Sheppard, E. Soloway (Eds.), *Empirical Studies of Programmers: Second Workshop* (pp. 100–113), NJ: Ablex.

Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3), (pp. 372–422).

Romero, P., du Boulay, B., Cox, R., Lutz, R. (2003). Java debugging strategies in multi-representational environments. In *Proceedings of the 15th Annual Workshop of the Psychology of Programming Interest Group* (pp. 421–434).

Romero, P., Lutz, R., Cox, R., du Boulay, B. (2002). Co-ordination of multiple external representations during Java program debugging. In *Proceedings of the Empirical Studies of Programmers symposium of the IEEE Human Centric Computing Languages and Environments Symposia 2002* (pp. 207–214).

Sajaniemi, J., Kuittinen, M. (2003). Program animation based on the roles of variables. In *Proceedings of the 2003 ACM symposium on Software visualization* (pp. 7–16), NY: ACM.

Shneiderman, B., and R. Mayer. (1979) Syntactic Semantic Interactions in Programmer Behavior: A Model and Experimental Results. *International Journal of Computer & Information Sciences*, 8(3), (pp. 219–238).

Sibert, L. E., Jacob, R. J. K. (2000). Evaluation of eye gaze interaction. In *Proceedings of CHI 2000* (pp. 281–288), NY: ACM.

Zeller, A. and Lütkehaus, D. (1996). DDD, A Free Graphical Front-End for UNIX Debuggers. *ACM SIGPLAN Notices*, 31(1), (pp. 22–27).

P4.

4

Bednarik, R., Randolph, J.: **Studying Cognitive Processes in Program Comprehension: Levels of Analysis of Sparse Eye-Tracking Data**. In Hammoud, R. (ed): Passive Eye Monitoring: for Safety, Security, Communications, Medical and Web Applications. 2008, Springer.

Studying Cognitive Processes in Program Comprehension: Levels of Analysis of Sparse Eye-Tracking Data

Roman Bednarik and Justus Randolph

Department of Computer Science and Statistics, University of Joensuu, Finland.
roman.bednarik@joensuu.fi

Analysis of eye-tracking data and their relation to underlying cognitive processes is one of the principal challenges of eye-tracking research. This chapter addresses the problem in the domain of studies of programming where the data can be sparse. In those situations, the traditional parametric methods to analyze the data might fail. We review previous recent approaches to the analysis, and we present solutions for the analysis of sparse data and illustrate their benefits for eye-tracking research.

1.1 Introduction

Computer programming is a cognitively demanding task [15, 10]. Programmers have to constantly apply their knowledge and skills to acquire and maintain a mental representation of a program. Modern programming environments (sometimes called integrated development environments (IDE)) often present, in several adjacent windows, the information related to the actual program. These windows often contain a variety of representations of the program (e.g. the program text or a visualization of program variables), all of which need to be taken into consideration by the programmer. Investigating a programmer's visual attention to these different representations can lead to insights about how programmers acquire the skill of program comprehension, which is considered to be the crux of successful computer programming. Research questions related to the role of visual attention during program comprehension might focus, for instance, on whether visual attention patterns differentiate good and poor comprehenders, or what are the specific features of a program text that skilled programmers attend while debugging a flawed program.

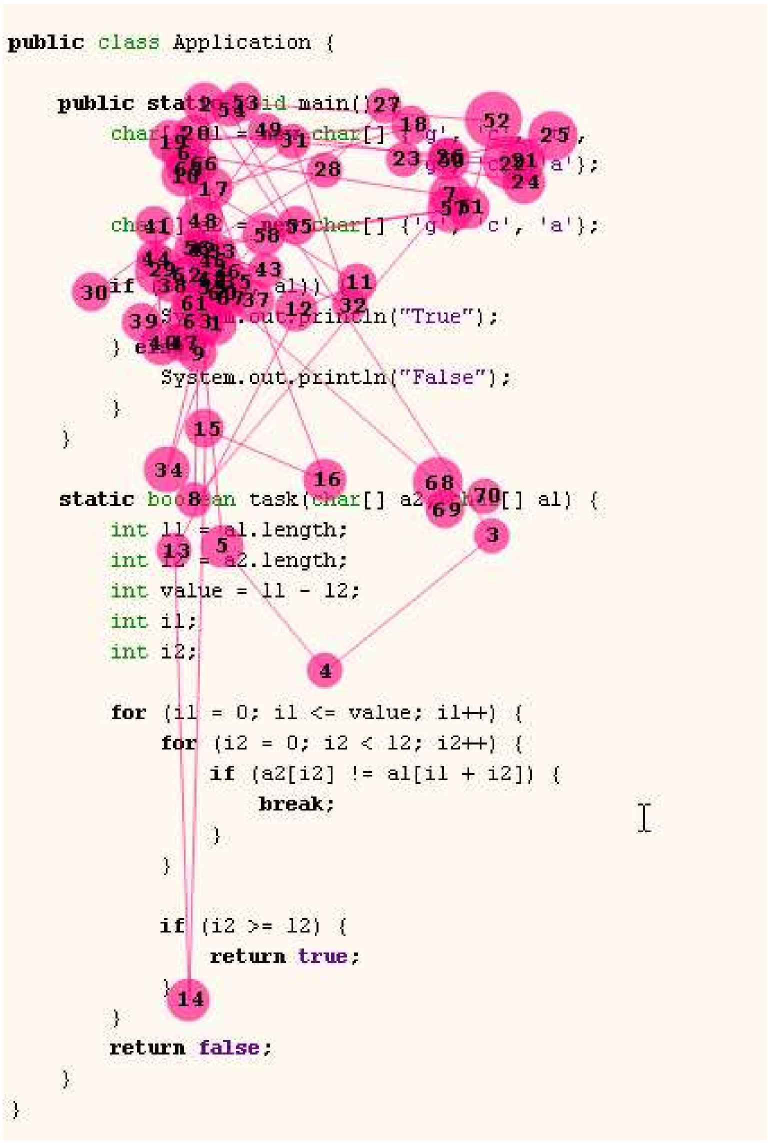
Eye-movement tracking has been employed across a variety of domains to capture patterns of visual attention [11]. Among the many domains, studies of visual attention during reading is, probably, the domain that has benefited the most from eye-tracking technology (c.f. [23] for a review). In addition to studies of reading, other domains have benefited substantially from adopting eye-tracking methods to investigate visual attention, including studies of driving (e.g. [27, 24]) or studies of the usability of computer interfaces (e.g. [13, 14, 8]).

Despite the many benefits of eye-tracking, there are also several disadvantages and challenges to its application. Although the technical challenges to eye-tracking research are rapidly being overcome, there are several methodological challenges that still stand in the way. For example, Jacob and Karn [16] argue that the analysis of eye-tracking data is one of the principal challenges for eye-tracking studies. This indeed seems to be the case in the domain of studying visual attention in programming visual attention in programming, as we discuss in this chapter.

A typical course of eye-movements during a program comprehension session is illustrated in Figure 1.1. The gaze-plot in Figure 1.1 displays approximately one minute from the beginning of a Java program comprehension task; fixations, numbered consecutively, are displayed as circles and their diameter corresponds with the duration. In this particular case, the programmers were instructed to comprehend the program as best as they could with the purpose of writing a comprehension summary. From the figure it can be observed that the programmer first briefly overviewed the whole source code and then carefully investigated the first part of the program.

In many laboratory eye-tracking program comprehension studies, participants are free to select which programming strategy and approach they engage in, to emulate what happens in real life programming tasks. The result is that if a certain programming activity is the phenomenon of interest, only the data of participants who willfully chose to do that certain programming procedure can be used for analysis. So for example if only 10% of participants tend to do a certain programming task, and 30 cases are needed to ensure the proper power for parametric tests, then 300 participants would be needed to get a sufficient amount of data. However, given the time and expense involved in eye-tracking studies, collecting data from this large of a number of cases is simply not feasible. The consequence, along with the fact that portions of data might be missing due to technical difficulties, is that eye-tracking data sets in program comprehension tend to be sparse.

The sparseness of the data sets leads to serious problems in applying conventional parametric statistical methods. In this paper we discuss two "back-to-basics" approaches that we have found to have been helpful in analyzing the sparse data sets generated from eye-tracking studies of program comprehension, especially those analyses dealing with the identification of trends. Those approaches are conducting visual analyses of data (in the single-case tradition) or conducting a binomial test. We admit that these methods are just a few of many suitable ways to analyze sparse data sets. The utility of the methods presented here is grounded in their parsimony and ease of conceptual understanding. They constitute what we believe to be a "minimally



```
public class Application {

    public static void main() {
        char[] a1 = new char[] {'g', 'c', 'g',
                'c', 'a'};

        char[] a2 = new char[] {'g', 'c', 'a'};
        if (task(a2, a1))
            System.out.println("True");
        } else
            System.out.println("False");
        }
    }

    static boolean task(char[] a2, char[] a1) {
        int l1 = a1.length;
        int l2 = a2.length;
        int value = l1 - l2;
        int i1;
        int i2;

        for (i1 = 0; i1 <= value; i1++) {
            for (i2 = 0; i2 < l2; i2++) {
                if (a2[i2] != a1[i1 + i2]) {
                    break;
                }
            }

            if (i2 >= l2) {
                return true;
            }
        }
        return false;
    }
}
```

Fig. 1.1. Gaze-plot of one minute of eye-movement data during program comprehension.

sufficient analysis" [1] of the types of sparse data sets common to eye-tracking studies of program comprehension. Although these approaches were applied to eye-tracking program comprehension data sets, there is no reason that they would not generalize to other domains where eye-tracking data sets tend to be sparse and have few cases. For example, studies of usability, as we also discuss later, are conducted using similar research settings and designs as studies of program comprehension. In both cases, users work under minimal restrictions, their tasks are as natural as possible, and the number of participants is similar.

This chapter is organized as follows. Before we explain the approaches to analysis that we have taken, we first give an overview of the methods employed in visual attention research in programming and then provide some examples. Next we present information about the visual analysis of eye-tracking data and about conducting a binomial test to identify trends.

1.2 Related work

1.2.1 Techniques to capture visual attention during programming activities

At the moment, there are three types of techniques that have been employed to estimate the current location of user's focus of visual attention. The first category includes systems based on tracking the direction of user's gaze (i.e. eye-tracking systems), the second category includes systems based on focus windows (where all of the screen is blurred except for one unrestricted window), and the third category includes systems that record verbal protocols [12] as users problem-solve. In this chapter, we focus on the analysis of data that result from the eye-tracking and focus window systems, and do not further consider the analysis of verbal protocols.

Most modern eye-tracking systems are based on video and image processing of eye movements; those systems capture the reflections from infrared emitters pointed at the eye. Typically, the vendors of eye-tracking systems also provide a software package that facilitates the recording of gaze location and the analysis of the resulting stream. An extensive treatment of the technical issues involved in eye-tracking can be found in [11].

The Focus Window Technique (FWT) is an alternative method for tracking visual attention. The technique was designed to reduce the technical problems with eye-tracking. In this method, the screen is blurred except for a certain section of the screen. The Restricted Focus Viewer (RFV) [17] is a system based on the FWT. In the RFV technique, a user controls the location of the only focused area with a computer mouse. The RFV produces a stream of data that is similar to that of the eye-tracker: the resulting file contains time-stamped data that contains the location of the focused spot, mouse clicks, key presses, and other events. The RFV or systems based on it have been applied in several studies (e.g. [26]); Bednarik and Tukiainen [7], however, showed that the focus window technique, when employed to track visual attention during debugging, interferes with the natural strategies of programmers. Jones and Mewhort [18] also showed that the FWT has to be properly calibrated to prevent the user from exploiting information that should have been made inaccessible through blurring.

1.2.2 Studying visual attention in programming behavior using eye-tracking

Normally, computer programmers work with graphical, highly interactive development systems. These systems present program-related information in multiple windows that provide different representations of the program and its execution. While programming, programmers have to coordinate these windows to build and maintain a coherent model of the program. It is not clear, however, what the role of visual attention is in this process. Studies in the domain of the psychology of programming therefore attempt to gain insights into visual attention patterns during program maintenance tasks and during interaction with multi-representational development environments.

In this section, we describe how studies of programming behavior and visual attention are usually conducted, including information about the number of participants, the materials, eye-tracking apparatus used, and the research designs that are typically involved. We came about this information by reviewing a purposive sample of previous studies that reported an empirical investigation of the role of visual attention in some programming-related task. Although we did not conduct a systematic search of the literature, based on our familiarity with the field and because the field is a small one, we are confident that these studies, if not a comprehensive sample, are at least representative of the situations and methods of analysis being currently applied in visual attention studies of programming. One of the focuses of the review was on how the analyses of the data had been performed.

Table 1.1 summarizes 11 studies of programming that had incorporated a visual-attention-tracking technique. It presents the number of participants involved, the type of experimental design, the type of analysis, and duration of task(s) from which the eye-tracking data were aggregated into a measure and analyzed. If the maximal duration of a single session was reported instead of the actual mean duration, we used the maximal duration. The studies of Romero and colleagues [26, 25] employed the RFV, while the other studies used an eye-tracking device to record visual attention.

In terms of the number of experimental participants, in studies using an eye-tracker there were 13 participants on average; however, in studies using an RFV there were about 45 participants on average. The average number of participants in the eye-tracking studies reviewed here was close to the number that has been found in other reviews. For example, the average number of participants in the review of 24 eye-tracking studies of usability reported by Jacob and Karn [16] was 14.6. In our sample, the average duration of a session from which the visual attention data were analyzed, if reported, was about 13.5 minutes.

Table 1.1. Summary of 11 empirical studies of programming that employed eye-tracking

Authors (year)	**Subjects**	**Design**	**Analysis method**	**Duration of condition (min)**
Crosby and Stelovsky (1990)	19	between-subject	difference in means, visual	-
Romero et al. (2002)	49	mixed	ANOVA	10.0
Romero et al. (2003)	42	within-subject	ANOVA	10.0
Bednarik and Tukiainen (2004)	10	within-subject	difference in means	10.0
Nevalainen and Sajaniemi (2004)	12	within-subject	difference in means	8.3
Bednarik et al. (2005, 2006a)	16	between-subject	ANOVA	9.8 - 17.6
Nevalainen and Sajaniemi (2005)	12	within-subject	ANOVA	26.3
Nevalainen and Sajaniemi (2006)	16	between-subject	ANOVA	12.5 - 19.2
Bednarik et al. (2006b)	16	between-subject	correlations	-
Uwano et al. (2006)	5	between-subject	visual	-

As Table 1.1 shows, between-group or a within-group experimental designs were common, mixed designs were rare, and no study used a single-case design. We also found that parametric statistical methods were the norm. Very few studies had taken a visual approach to the analysis of the visual attention.

In summary, based on our sample, a typical eye-tracking study of behavioral aspects of programming involves about ten to sixteen participants. Participants are given a task, most frequently to comprehend or to debug a relatively short program. The interface of the stimulus environment is divided into several (usually large) nonoverlapping areas of interest, and eye-tracking data is recorded and aggregated with respect to these areas. Researchers then often conduct some type of parametric statistical procedure to compare the means of the eye-tracking measures between the areas of interest.

We believe that although the parametric approaches to the analysis of eye-tracking data are useful in certain cases, nonparametric procedures and visual analyses are important as well. For example, only two studies in our sample conducted a visual analysis of the resulting data. In the remaining studies, typically, the eye-tracking data is aggregated into a single measure across the duration of a session. For example, the measure of relative fixation counts on a certain area is computed as a proportion of the number of fixations anywhere on that area (e.g. on the code window) to the all fixations registered during the session. In programming tasks, however, the changes in visual attention can also be miniature, compared to the focus of the analyses of data from the larger areas. Crosby and Stelovsky [9], for example, showed differences in targeting visual attention on comments–therefore on the smaller areas within the program text. When eye-tracking measures are aggregated as described above, the miniature differences might disappear. An eye-tracking researcher can, of course, decide to increase the number of areas and design the matrix so that it still covers the screen but the areas are smaller (see for example Bednarik et al. [4]). In the case of increasing the number of areas to be analyzed, however, the resulting data become sparse, because even more of the areas (and cells with measures) become empty.

To deal with the challenges described above, we suggest two alternative solutions for the analysis of eye-tracking data: visually analyzing eye-tracking data at two levels of detail, and conducting a binomial test to investigate trends in sparse eye-tracking data.

1.3 Minimally sufficient analyses of eye tracking data sets that are sparse and have few cases

As we mentioned before, one unfortunate characteristic of eye-tracking data sets is that they are often sparse (i.e., missing a large amount of data) or have few cases. The sparseness of the data often results from errors in eye tracking measurements (such as loss of gaze due to the participant wearing eye-glasses or contact lenses, or head movements during recording) or from specialized types of research situations in which eye-tracking participants can choose how many measurements they will participate in. The low number of cases results from the intensity of eye-tracking measurements. For example, in studies of programming or in usability research the duration of a condition during which data is recorded is normally between ten minutes to a half an hour per condition. Because it takes approximately ten times more time to analyze and interpret the resulting data, it is no surprise that in most situations it is not practical to collect a very large number of measurements from a large number of participants.

Table 1.2 is a hypothetical example of an authentic eye-tracking data set that is sparse and has few cases. (The data are hypothetical but are based on an authentic data set. We chose to use a hypothetical data set because it better illustrated the properties and methodology that we discuss here than the authentic data set.) Table 1.2 shows the results of the ratio of fixation counts on two areas of interest from 12 participants over four possible measurements.

In this hypothetical situation, participants in each measurement were asked to do a programming task; each task was similar to the others. Participants were allowed to repeat similar tasks up to four times so that they could master those programming tasks. Some participants, such as participants J, K, and L decided that they only needed to do the task one or two times.

Table 1.2. Hypothetical eye-tracking data set

	Fixation Ratio			
Participant	**Measurement 1**	**Measurement 2**	**Measurement 3**	**Measurement 4**
A	1.3	1.5	1.6	1.7
B	2.3	2.4	-	-
C	2.5	2.6	2.7	-
D	1.6	1.7	-	-
E	1.9	2.0	2.1	-
F	1.5	1.6	1.7	1.8
G	2.0	2.1	1.9	-
H	1.8	1.5	1.6	1.7
I	1.9	2.0	-	-
J	3.0	-	-	-
K	2.7	-	-	-
L	2.8	-	-	-

Other participants, like participants A, C, and E, decided that they needed to do three or four programming tasks to achieve mastery. The optional repetition of programming tasks was an inherent attribute of the research question: Do the fixation ratio times show an increase as optional programming tasks are repeated? The optional aspect in that research question is what causes the sparseness of the data; some participants will only complete one measurement, other participants will complete two or more measurements. The low number of measurements has to do with the fact that it took so much time and resources to complete a single measurement.

The question, then, is how does one analyze data sets such as the one presented in Table 1.2. Traditionally, a repeated-measures analysis (i.e., an all-within-design) would be used for this type of design. But when using that approach on this particular data set, because it is sparse, the power would be very low after excluding cases with missing data. In this case, the statistical power would have been only 0.12. Replacing the missing data would be questionable in this case because there are missing data in about 38% of the cells.

Instead of treating the data as repeated measures, one might argue that the data for each attempt could be treated as independent and means for each measure could be compared as if one were doing a one-way ANOVA analysis (i.e., an all between analysis) with four groups. The groups would be Measurement 1 (n=12), Measurement 2 (n=9), Measurement 3 (n=6), and Measurement 4 (n=3). However, this would seriously undermine the statistical assumptions involved in this kind of parametric analysis (especially, treating dependent measures as independent measures.) Even if one were still able to conclude that the assumption violation were acceptable, the statistical power would still be poor–in this case it would be 0.17.

Also, the choice between a repeated-measures analysis (all-within design) with three full cases (i.e., participants A, F, and G) and a one-way ANOVA type analysis with four groups and 30 total data points makes a striking difference in what the data show. In Figure 1.2 the means across each measurement for the four full cases are taken, as would happen in a repeated-measures analysis without the replacement of data. Figure 1.2 shows that there is an upward trend over time. In Figure 1.3, the means within each measurement of all the cases are taken, as would happen if a one way ANOVA type analysis were used. Contrary to Figure 1.1, Figure 1.3 shows that there is a downward trend.

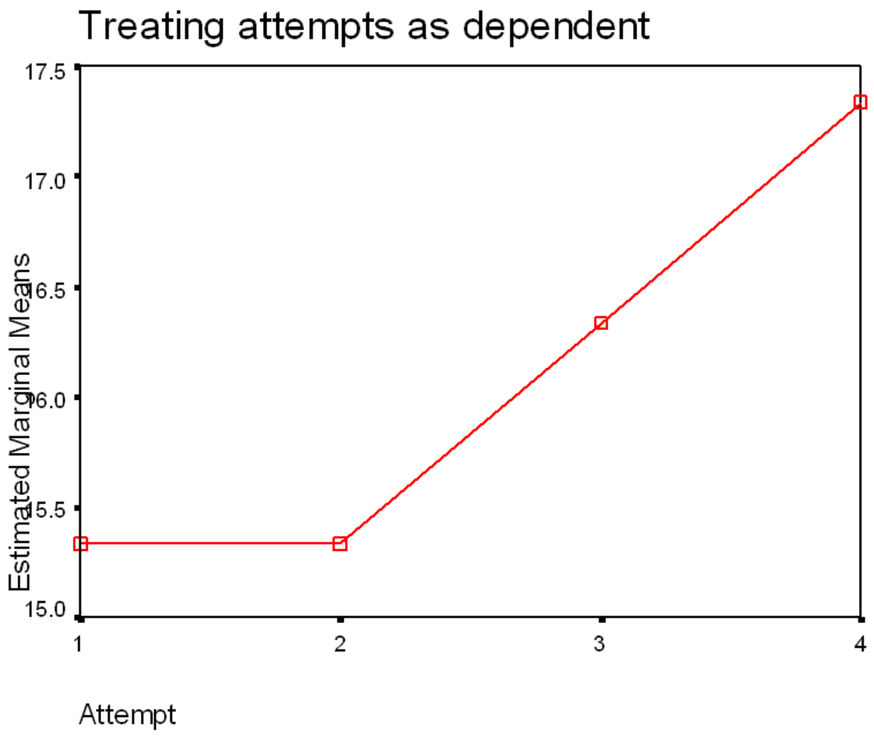


Fig. 1.2. Plot of measurement means of the four complete cases.

Because of the low power and violation of statistical assumptions when using parametric types of analyses for the analysis of sparse data sets with

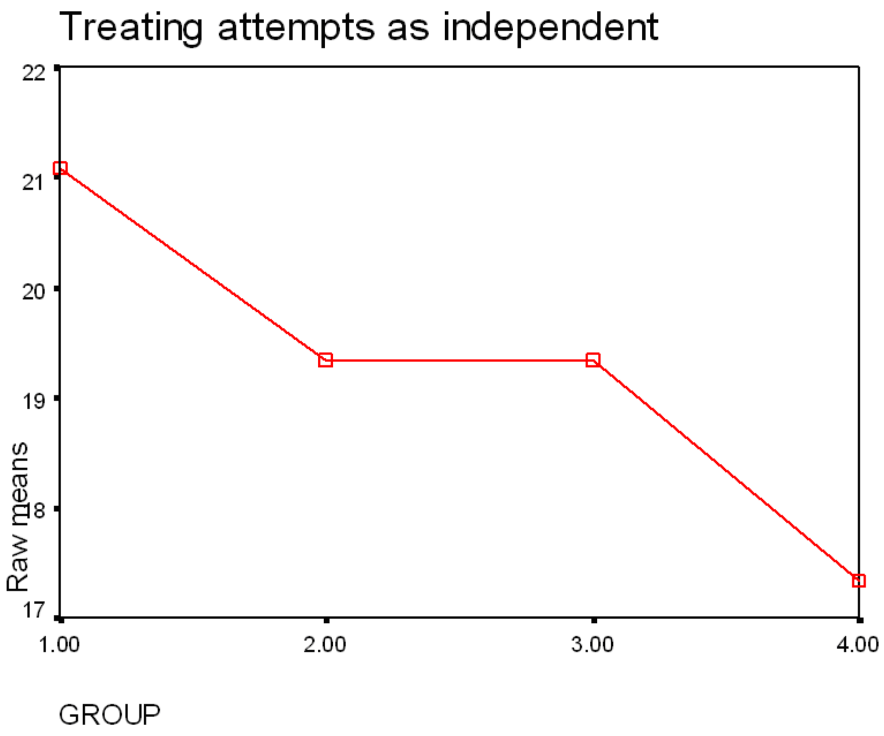


Fig. 1.3. Plot of measurement means of all cases.

few cases, we have found that the most reasonable solution is to get back to the basics and do minimally sufficient analyses. The minimally sufficient analytic solutions we have found for these types of data sets are visually inspecting the data and using a binomial test. (Using the Page test [22] is another parsimonious solution, but since it is much more complex than using the binomial distribution or visual analyses, we do not present it here.) How one would use those two approaches (i.e., visual analysis and the binomial test) in terms of the data set presented in Table 1.2 is presented below.

1.3.1 Visual analysis of the eye-tracking data

Simply visually inspecting a graph of the data can provide a wealth of information that would be overlooked by looking at the raw numbers in a data set. Figure 1.4 shows the data when they are arranged in a series and plotted by attempt and fixation ratio. First it appears that there is an upward trend–the fixation ratio increases each subsequent trial. Also, Figure 1.4 shows that participants who only did one trial were those with the highest fixation ratios to begin with and that the participants who did the most optional programming tasks were the ones with the lowest fixation ratios. The visual analysis of single cases approach presented here could also be extended to include types of research studies other than trend studies. For example, one could use an ABA design where the A phase is a control condition and the B phase is the treatment condition, or any of the other designs successfully used in single-case research (see [2] for more information on the variety of single-case designs).

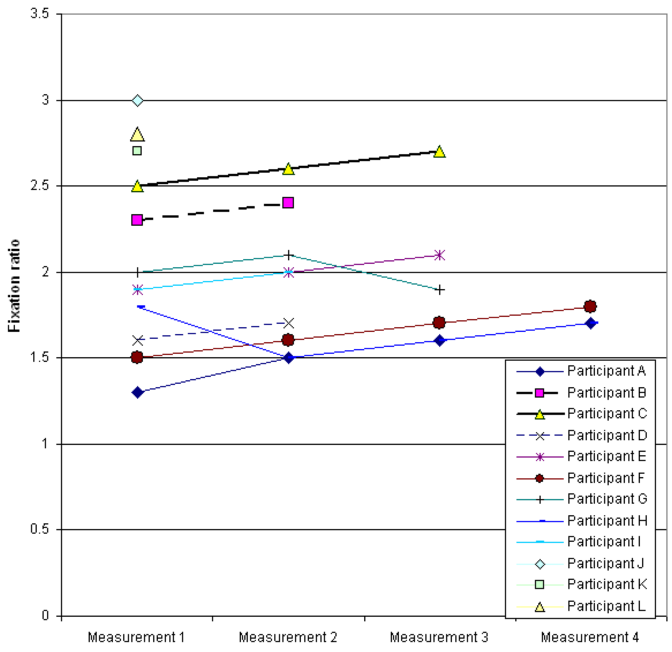


Fig. 1.4. Line graph of fixation ratios by measurement.

While the previous approach operates with the eye-tracking data on the measure level, there is, however, yet another, finer level of detail that a researcher can use in order to analyze the eye-tracking data. Single fixations can be plotted against a time line, to discover the subtle differences and patterns in visual attention. Figure 1.5 shows about fifteen seconds of fixation data from two participants during comprehension of a program, where the vertical axis displays the Y-coordinate (the vertical coordinate of the gaze point). Thus, when aligned side-by-side with the window containing the source code, the graph represents the approximate lines of code and order in which the participants visually attended to them. The horizontal dashed line shown in Figure

1.5 (with the Y-coordinate of 238) represents the long-term average of both of the participants' visual attention on the Y-coordinate. It is worth mentioning that parametric statistical analyses would use these same averages to compare the participants. From the course of the visual attention lines, however, we can observe that the two participants markedly differed in their comprehension strategies and their actual visual attention focus had only rarely been found near the average line.

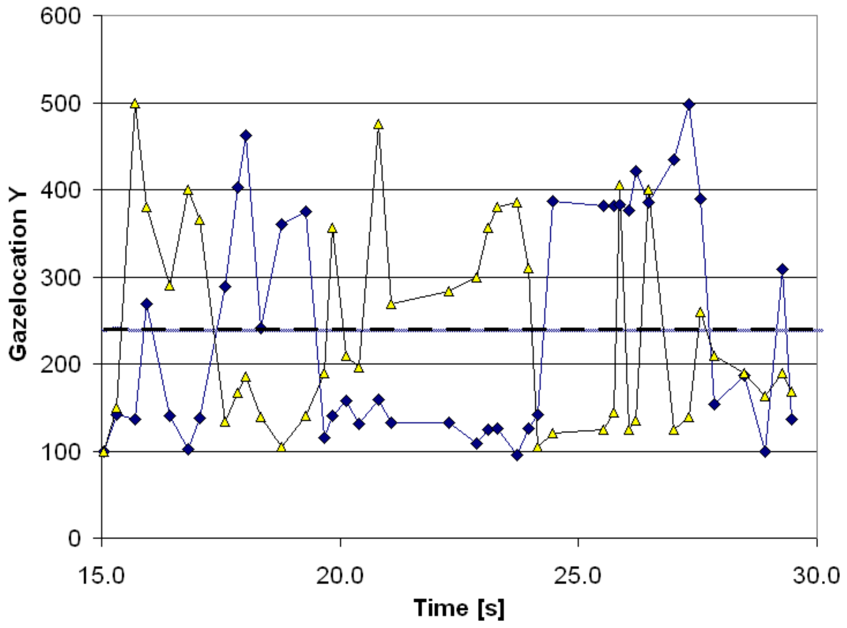


Fig. 1.5. A 15-second segment of the Y-coordinate of the visual attention focus of two participants comprehending a program.

1.3.2 Analyzing trends of the sparse eye-tracking data

If the visual inspection of single-case data were not convincing enough, the binomial distribution can be used to easily determine the probability of getting a certain number of increases between measures given chance. If there were no trends in the data set, one would expect about the same number of increases and decreases to occur, if there were an increasing trend one would see a greater number of increases than decreases, and if there were a decreasing trend one would see a smaller number of increases than decreases. When the outcomes are only either a success or failure, the binomial distribution can be used to determine the probability that out of a given number of trials a certain number of successes (or failures) will occur. For example, the binomial distribution can be used to determine the probability that, given a fair coin, heads will come up 5 out of 5 times that the coin is tossed (the probability is a little bit over 3/100). The binomial distribution is given in Equation 1.1. In Equation 1.1, p is the probability of success, N is the number trials, and x is the number of successful trials out of N trials. probability, such as online calculators or built-in procedures in tools such as Matlab or SPSS, that can be used to automatically determine a binomial probability.

$$P(X \geq x) = \sum_{i=x}^{N} P(X = i) = \sum_{i=x}^{N} \frac{N!}{i!(N-i)!}(p)^i(1-p)^{(N-i)} \tag{1.1}$$

To use the binomial distribution to find the probability of getting the number of increases that were observed, one must convert the data set into binomial trials. Table 1.3 shows what the data set in Table 1.2 would look like if it were converted into binomial trials, where i indicates that there was an increase between earlier and later measurement and where d indicates that there was a decrease between earlier and later measurements. The data in Table 1.3 show that there were 30 trials ($N = 30$), 22 increases ($i = 22$), and we assume that the probability of success is .50 ($p = .50$). Using Equation 1.1, the cumulative probability of getting 22 successes or more out of 30 trials where the probability of success is .50 is .008. (To find the cumulative probability of 22 or more successes out of thirty one has to add the probabilities of getting 22 of 30, 23 of 30, . . . , 30 out 30.) Figure 1.6 shows the binomial distribution for getting 22 out of 30 successful trials when the probability of success is .50 and the standard deviation is 2.74. As shown in Figure 1.6, the expected number of successful trials given chance is 15. The standard deviation is 2.74. The observed number of trials, 22, was much greater; it was 2.56 standard deviations greater than the expected number of successes. In short, the probability of getting 22 out of 30 increases by chance in this case was very small; therefore, one can conclude that there was an increasing trend in the data set.

There are, however, also variations to the presented approach of transforming measures to trials, each of those having a set of advantages and disadvantages. For instance, one approach to transforming the data set into trials is to use differences only between consecutive measures (e.g. 1-2, 2-3, 3-4). In that case the method can discover a prevailing number of increases or decreases, but not an overall trend in the measures across all possible combinations. Our method of transformation of measures into trials takes each possible pair of measures within a subject and transforms those into trials. That is, for example the pair 1-4 is also included as an increase if the fourth trial had been greater than the first trial, or as a decrease, otherwise.

Table 1.3. Hypothetical data set converted to binary trials, i = increase, d = decrease

Case	1-2	1-3	1-4	2-3	2-4	3-4	Total increases	Possible increases
A	i	i	i	i	i	i	6	6
B	i	-	-	-	-	-	1	1
C	i	i	-	i	-	-	3	3
D	i	-	-	-	-	-	1	1
E	i	i	-	i	-	-	3	3
F	i	i	i	i	i	i	6	6
G	i	d	-	d	-	-	1	3
H	d	d	d	d	d	d	0	6
I	i	-	-	-	-	-	1	1
J	-	-	-	-	-	-	0	0
K	-	-	-	-	-	-	0	0
L	-	-	-	-	-	-	0	0
						Totals	22	30

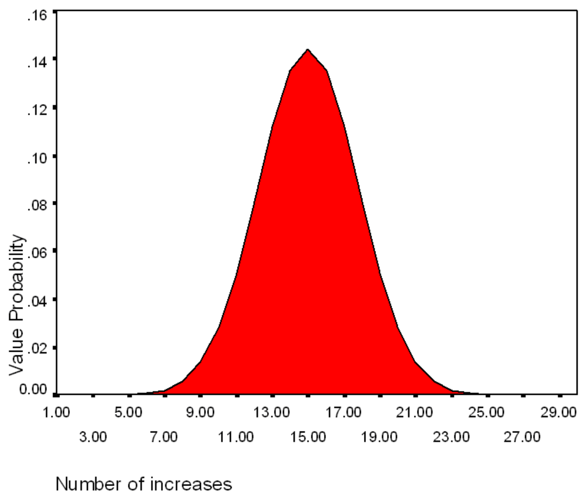


Fig. 1.6. Binomial distribution for 30 trials when probability of success is .50. The expected value is 15 and the standard deviation is 2.74. The cumulative probability of getting 22 or more increases out of thirty is 0.008.

As a result, while identifying the overall trends, our method leads to a greater number of trials than the consecutive measure transformation method.

Therefore, the p-value using the all possible combinations method will also differ from the p-value using the consecutive measure method discussed above. Our method has the disadvantage of not having mathematically independent trials. For example, if there is an increase between Measures 1 and 2 and an increase between Measures 2 and 3, then, by definition, there will be an increase between Measures 1 and 3.

We admit that there are also many general disadvantages to using the binomial distribution to analyze these types of data. Namely, the conversion of continuous values into binary values loses a significant amount of information. Using the binomial approach for example, one cannot comment on how great the increases were, but only one on the probability that a certain amount of increases would have occurred given chance. We also admit that there are other, more-complicated nonparametric procedures that could be used in this case to identify trends (like the Page test [22]). However, we agree with the American Psychological Association [1] and the American Psychological Association's Task Force on Statistical Inference Testing [29] on the importance of conducting a minimally sufficient analysis. In the case of data sets that are sparse and have few cases, we propose that at least two of the candidates for a minimally sufficient analysis is through the visual inspection of data and using basic nonparametric tools, such as the binomial distribution.

1.4 Conclusions

While eye-tracking data have some properties that make them difficult to analyze using conventional techniques, they also have qualities that enable a researcher to easily create a chronological narrative from the data. Our review showed, however, that researchers employing eye-tracking in studies of programming tend to analyze eye-tracking data at the most aggregated levels of analysis and seem to ignore analyzing the data through visual inspection.

In summary, we have shown one method for statistically identifying trends in eye-tracking datasets that are sparse. In addition, we have also proposed to visually inspect eye-tracking data either by examining each participant's aggregated scores across measures or by visually inspecting a graph of an individual's data over time and plotted against given areas of the screen. The visual analysis methods presented here can reveal intricate and subtle patterns of individual visual attention behavior that, when woven together, can provide a more detailed narrative than when parametric procedures are used alone.

References

1. American Psychological Association. *Publication manual of the American Psychological Association (5th Ed.).* Washington, D.C, USA, 2002.

2. D. H. Barlow and M. Hersen. *Single Case Experimental Designs (2nd Ed.)*. Pergamon, New York, USA, 1992.
3. R. Bednarik, N. Myller, E. Sutinen, and M. Tukiainen. Effects of experience on gaze behaviour during program animation. In *Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05)*, pages 49–61, Brighton, UK, 2005.
4. R. Bednarik, N. Myller, E. Sutinen, and M. Tukiainen. Analyzing Individual Differences in Program Comprehension with Rich Data. *Technology, Instruction, Cognition and Learning*, 3(3-4):205–232, 2006.
5. R. Bednarik, N. Myller, E. Sutinen, and M. Tukiainen. Program visualization: Comparing eye-tracking patterns with comprehension summaries and performance. In *Proceedings of the 18th Annual Psychology of Programming Interest Group Workshop (PPIG'06)*, pages 68–82, Brighton, UK, 2006.
6. R. Bednarik and M. Tukiainen. Visual attention and representation switching in java program debugging: A study using eye movement tracking. In *Proceedings of the 16th Annual Psychology of Programming Interest Group Workshop (PPIG'04)*, pages 159–169, Carlow, Ireland, 2004.
7. R. Bednarik and M. Tukiainen. Validating the Restricted Focus Viewer: A Study Using Eye-Movement Tracking. *Behavior Research Methods*, to appear.
8. L. Cowen, L. J. Ball, and J. Delin. An eye-movement analysis of web-page usability. In X. Faulkner, J. Finlay, and F. Détienne, editors, *People and Computers XVI: Memorable yet Invisible: Proceedings of HCI 2002*. Springer-Verlag Ltd, 2002.
9. M. E. Crosby and J. Stelovsky. How Do We Read Algorithms? A Case Study. *IEEE Computer*, 23(1):24–35, 1990.
10. F. Détienne. *Software Design: Cognitive Aspects*. Springer, November 2001.
11. A. T. Duchowski. *Eye Tracking Methodology: Theory & Practice*. Springer-Verlag, Inc., London, UK, 2003.
12. K. Anders Ericsson and Herbert A. Simon. *Protocol analysis: Verbal reports as data*. MIT Press, Cambridge, MA, 1984.
13. J. Goldberg and X. P. Kotval. Computer Interface Evaluation Using Eye Movements: Methods and Constructs. *International Journal of Industrial Ergonomics*, 24:631–645, 1999.
14. J. Goldberg and A. Wichansky. Eye Tracking in Usability Evaluation: A Practitioner's Guide. In J. Hyn, R. Radach, and H. Deubel, editors, *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, pages pp. 493–516. Elsevier Science, 2003.
15. J. M. Hoc, T. R. G. Green, R. Samurcay, and D. J. Gilmore. *The Psychology of Programming*. Academic Press, 1990.
16. R. J. K. Jacob and K. S. Karn. Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises (Section Commentary). In J. Hyn, R. Radach, and H. Deubel, editors, *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, pages pp. 573–605. Elsevier Science, 2003.
17. A. R. Jansen, A. F. Blackwell, and K. Marriott. A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, and Computers*, 35(1):57–69, 2003.
18. M. N. Jones and D. J. K. Mewhort. Tracking attention with the focus-window technique; the information filter must be calibrated. *Behavior Research Methods, Instruments, and Computers*, 36(2):270–276, 2004.

19. S. Nevalainen and J. Sajaniemi. Comparison of three eye tracking devices in psychology of programming research. In *Proceedings of the 16th Annual Psychology of Programming Interest Group Workshop (PPIG'04)*, pages 151–158, Carlow, Ireland, 2004.
20. S. Nevalainen and J. Sajaniemi. 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)*, pages 77–91, Brighton, UK, 2005.
21. S. Nevalainen and J. Sajaniemi. An experiment on short-term effects of animated versus static visualization of operations on program perception. In *ICER '06: Proceedings of the 2006 international workshop on computing education research*, pages 7–16, New York, NY, USA, 2006. ACM Press.
22. E. B. Page. Ordered hypotheses for multiple treatments: A significance test for linear ranks. *Journal of the American Statistical Association*, 58:206–230, 1963.
23. K. Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124:372–422, 1998.
24. S. D. Rogers, E. E. Kadar, and A. Costall. Drivers' gaze patterns in braking from three different approaches to a crash barrier. *Ecological Psychology*, 17(1):39–53, 2005.
25. P. Romero, B. du Boulay, R. Lutz, and R. Cox. 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, 2003. IEEE Computer Society.
26. P. Romero, R. Lutz, R. Cox, and B. du Boulay. 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)*, page 207, Washington, DC, USA, 2002. IEEE Computer Society.
27. G. Underwood, P. Chapman, N. Brocklehurst, J. Underwood, and D. Crundall. Visual attention while driving: Sequences of eye fixations made by experienced and novice drivers. *Ergonomics*, 46(6):629–646, 2003.
28. H. Uwano, M. Nakamura, A. Monden, and K. Matsumoto. Analyzing individual performance of source code review using reviewers' eye movement. In *ETRA '06: Proceedings of the 2006 symposium on Eye tracking research & applications*, pages 133–140, New York, NY, USA, 2006. ACM Press.
29. L. Wilkinson and the Task Force on Statistical Inference. Statistical methods in psychology journals: Guidelines and explanations [Electronic version]. *American Psychologist*, 54:594–604, 1999.

P5.

5

Bednarik, R., Tukiainen, M.: **An Eye-tracking Methodology for Characterizing Program Comprehension.** In Proceedings the 2006 Symposium on Eye Tracking Research and Applications, ETRA 2006, March 27-29, San Diego, CA, USA, ACM Press, pp. 125 - 132.

An eye-tracking methodology for characterizing program comprehension processes

Roman Bednarik[*]
Department of Computer Science
University of Joensuu
PO Box 111, FI-80101, Finland

Markku Tukiainen[†]
Department of Computer Science
University of Joensuu
PO Box 111, FI-80101, Finland

Abstract

Program comprehension processes have previously been studied using methodologies such as think-aloud or comprehension summary analysis. Eye-tracking, however, has not been previously widely applied to studies of behavioral aspects of programming. We present a study in which program comprehension was investigated with a help of a remote eye-tracker. Novice and intermediate programmers used a program visualization tool to aid their comprehension while the location of fixations, fixation durations and attention switching between the areas of interest were recorded.

In this paper 1) we propose an approach how to investigate trends in repeated-measures sparse-data of few cases captured by an eye-tracker and 2) using this technique, we characterize the development of program comprehension strategies during dynamic program visualization with help of eye-movement data.

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

Keywords: eye-movement tracking methodology, psychology of programming, program comprehension, program visualization

1 Introduction

Program comprehension, the ability to understand programs written by others, is widely recognized as central to programming. Being also a cognitively complex skill of acquiring mental models of structure and function of a program, program comprehension has been for many years a field of need to develop and apply methods and techniques to effectively capture and analyze the involved mental processes. Although originally centered around (professional) computer-programmers developing computer programs, studies of programming strategies nowadays extend far beyond these borders [Blackwell 2002], both in terms of users and application domains.

Previous research in the domain has established a solid body of knowledge about the comprehension models and strategies employed in comprehension, expert and novice differences or comprehension outcome analysis [Good and Brna 2004]. Other studies concentrated on developing tools to aid program understanding, especially by means of visual representations of program execution, e.g. [Moreno et al. 2004; Sajaniemi and Kuittinen 2003]. Surprisingly, given the importance of identifying the cognitive processes involved in program comprehension, little has been done by applying visual attention tracking systems such as eye-movement tracking. Instead, investigations have been mostly based on verbal-protocols, a well established - and probably the most popular - method used to capture the thought-processing.

To safely apply eye-movement tracking to study the behavioral aspects of programming, however, requires to develop a methodological framework. This study starts to build such a framework. As eye-tracking has not been previously applied in the domains of program comprehension and visualization, despite clearly presenting a strong potential, our motivation is to create a methodological framework that would allow to apply the eye-tracking and provide a way to analyze the data. An obvious solution to the latter problem would be to use some variation of repeated measures designs and analysis. However, as seen from previous studies (e.g. [Bednarik et al. 2005]), treating the data as average values over a longer period - without considering the actual underlying task- might blur individual differences in between-participant and within-participant trials. In addition, the comprehension process with program visualization is dynamic and gradual in its nature and cannot be effectively examined by studying long-term averages.

In this paper, we apply eye-movement tracking to a study of comprehension processes of programmers using a program visualization tool. To allow for as natural conditions as possible, the participants were not limited in the ways they used and interacted with the tool, particularly in replaying the visualizations. Therefore, although designed as a repeated-measures experiment, the resulting data have often a sparse structure. We present an approach to analyze the data and apply the method to characterize the strategies and behavior of programmers to coordinate multiple representations of a program during program comprehension.

2 Related Work

2.1 Capturing Mental Processing during Programming

Studies of cognitive processes involved in program comprehension tasks are central to our understanding of software maintenance and development [von Mayrhauser and Vans 1996]. Particularly popular techniques to capture the thought-processes are different variations and derivations of the think-aloud methodology, either concurrent or retrospective, since the seminal work of Ericsson and Simon [1984]. In the empirical studies of programmers and psychology of programming, the think-aloud methodology was applied in the pioneering experiments conducted by Soloway and others during 1980's and 1990's (e.g. [Letovsky 1986; Soloway et al. 1988; Littman et al. 1986; von Mayrhauser and Vans 1996] or recently

[*]e-mail: bednarik@cs.joensuu.fi
[†]e-mail: mtuki@cs.joensuu.fi

[Burkhardt et al. 2002; Ko and Uttl 2003] and many others). By analyzing verbal protocols, these and other studies attempted to get an access to cognitive processes and provide insights into what strategies programmers of various expertise take while undergoing the tasks of program comprehension.

To increase the validity of verbal reports, research practitioners often complement them with other concurrent behavioral data, such as direct observations, video or interaction protocols, as for example [von Mayrhauser and Vans 1996] or [Ko and Uttl 2003]. The approaches based on verbal reports, however, have been criticized widely, e.g. [Branch 2000; Nielsen et al. 2002; van den Haak et al. 2003], especially when used with complex tasks involving high cognitive load and requiring verbalizing visual information.

2.2 Previous Studies Using Visual Attention Tracking

Normally, programmers work within a computer-based (graphical) environments, such as program debugging or visualization tools. These environments often present some of the program representations in several adjacent windows and programmers have to coordinate these representations in order to construct a viable mental model. In these situations, when the problem-solving and formation of mental model is driven by visual information, such as during dynamic program visualization, it would be beneficial to capture and analyze the patterns of visual attention. As eye-movement data provide insight into attention allocation, it is also possible to infer underlying cognitive processes [Rayner 1998].

Understanding that opportunity, Romero et al. [2002] conducted a series of experiments that involved the Restricted Focus Viewer (RFV) [Jansen et al. 2003], a tool that links visual attention to a small fully-focused spot within an otherwise blurred stimulus. When an experimenter employs the RFV to track the visual attention, participants move the focused spot using a computer mouse to explore the visual representations in task. This approach, however, was shown to be questionable. Using a remote eye-tracker, Bednarik and Tukiainen [2004; 2005] replicated one of the experiments in which the RFV was employed. They suggested that the technique interferes with natural strategies involved in program debugging. Similarly as in the influential studies of Petre [1995], the visual representations in experiments of Romero et al. were static. Modern program visualization tools, however, often present the concepts in form of dynamic animations. The ecological validity of these experiments could be, therefore, questioned too.

Eye-tracking as a research methodology in studies of programmers has been previously applied to investigate how programmers read the code [Crosby and Stelovsky 1990]. Using an eye-tracker, patterns of programmers' visual attention were recorded while reading a binary search algorithm written in Pascal. Authors analyzed fixation times and number of fixations to reveal the strategies involved in reading source code. Crosby and Stelovsky shown, beside other findings, that while the subjects with greater experience paid attention to meaningful areas of source code and to complex statements, novice participants, on the other hand, visually attended comments and comparisons. Both groups paid least attention to the keywords and did not exhibit any difference in reading strategies.

Other studies using eye-movement tracking during program comprehension or debugging, such the one of Crosby and Stelovsky above, are infrequent and appeared only recently, e.g. [Bednarik and Tukiainen 2004; Bednarik and Tukiainen 2005; Nevalainen and Sajaniemi 2005]. Bednarik et al. [2006; 2005] reported that in terms of fixation counts and attention switching between main representations (code and graphical representation of execution) of a program during its animation, patterns of novice and expert programmers did not differ. An effect of experience was found, however, on overall strategies adopted to comprehend programs and on fixation durations.

Bednarik et al. [2005], however, approached the analysis of eye-movement data only from a long-term, global point of view: the data were treated as means over a whole comprehension session. To characterize the comprehension processes more completely, another, more detailed procedure has to be taken. In the present paper, we subdivide the comprehension process aided by a program visualization into meaningful pieces and study gradual changes in the related eye-movement patterns. By doing so, we believe to capture the changes in the role each of the representations take during the comprehension and to characterize the construction of the mental model of the comprehended program.

3 Experiment

The present experiment was conducted to discover whether there is any development in the way programmers visually attend the representations provided by a program visualization tool during program comprehension. In particular, we were interested in whether the role of different representations of a program changes in the course of time, as reflected in the gaze data. Ratios of fixation counts, attention switching, and fixation durations between the main representations of a program were analyzed. Since the present analysis reports the data recorded from an experiment previously described and reported elsewhere, the following sections might share some parts with the previous reports [Bednarik et al. 2006; Bednarik et al. 2005].

3.1 Participants

A total of eighteen participants were recruited from high-school students attending a university level programming course, undergraduate and graduate computer science students from the local university; each received a lunch ticket as an incentive. Data from two participants had to be discarded due to technical problems with eye-tracking. Therefore the eye-tracking results are based on data collected from 16 subjects (13 male, 3 female). The participants, according to their report, can be characterized as having, on average, 49.3 (SD = 54.1) months of experience with programming, 13.1 (SD = 12.8) months of experience with Java, 19.3 (SD = 29.5) months of experience with other programming language. Five participants had a previous experience with the program visualization tool used in this experiment, other two participants had a previous industrial experience. All participants had normal or corrected-to-normal vision, according to their own report.

For some parts of the analysis, participants were divided into two groups according to the number of animation runs they initiated during the experiment. The number of animation runs can be considered as a measure of experience and comprehension performance: those who animated the program once did not need the support of the visualization tool further but to confirm their hypothesis, while those running the visualization more times were in need of its help.

3.2 Materials

Three short Java programs, a factorial computation, recursive binary-search program and naïve string matching, were presented to the participants. The lengths of the programs (in lines of code) were 15, 34, and 38, respectively. Each of the programs generated only one line of output and did not require any user input. To make recognition of a program based on these surface features difficult, the names of the methods and variables were altered. In practice, method and variable names were made random and neural.

To visualize the Java programs, Jeliot 3 [Moreno et al. 2004], a novice-oriented program visualization tool, was used. The user interface of Jeliot 3 (Figure 1) consists of four separate areas: the Code is on the top left, the visualization is shown in the top right area (called Theater), the Control panel with VCR-like buttons to control the animation is on the bottom left, and the Output of the program is displayed in the bottom right panel. Moreover, the Theater area is further split into four discrete sections that detail a) the *method* calls, b) *expression evaluation*, c) *constants* and static objects, and d) instantiation of *objects and arrays*. In the present experiment, the specialized views of method-call-tree and history of execution provided by regular Jeliot 3 were disabled.

Jeliot 3 automatically visualizes execution of Java programs by demonstrating graphically the data and control flows, using an object-oriented approach. In a typical session, a user loads or write a program, compiles it and then selects Play or Step button to start the animation in a continuous or step-wise fashion, respectively. An animation step consists of highlighting a block of instructions, statement, assignment or expression in the Code and displaying their respective graphical visualization in the Theater area.

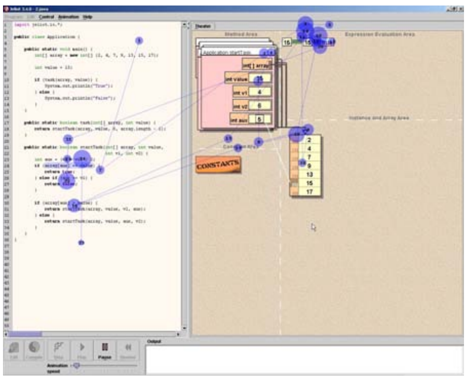


Figure 1: Program visualization tool used in the experiment with a representative scan-path superimposed.

3.3 Apparatus

As one of our requirements was to use a minimally invasive experimental setting, we used the remote Tobii ET-1750 (sampling at 50Hz) eye tracker that made no contact with participants (for setting see Figure 2). We used a single computer setup, in which operator's and participant's displays share same computer. The eye tracker is built into a TFT panel so no moving part is visible and no sound can be heard during recording. Interaction protocols (such as keystrokes and mouse clicks) were collected for all target programs, and audio and video streams were recorded for a whole session. To avoid involuntary fixations, a minimal duration of fixation for the algorithm processing the eye-data was set at 100ms. Seven static areas of interest (AOI) that matched with the seven main panels in the Jeliot 3 interface were defined: the code, the expression evaluation area, the method area, the instances area, the constants, the control, and the output area.

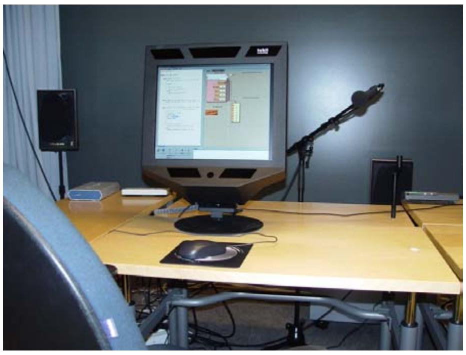


Figure 2: Experimental settings.

3.4 Procedure and Design

The experiment was conducted in a quiet usability laboratory. After becoming acquainted with the experiment and signing a consent form, participants were seated in an ordinary office chair near the experimenter and facing a 17" TFT display (resolution of 1024x768). Every participant then passed an automatic eye-tracking calibration. The calibration required the participants to follow sixteen shrinking points that appeared one by one across the display. If needed, the calibration was repeated in order to achieve the highest possible accuracy.

The window with program visualization tool was resized to occupy full-screen. Participants performed three sessions, each consisting of a comprehension phase using Jeliot 3 and a program summary writing phase. The target programs contained no errors, were always preloaded into Jeliot and compiled to demonstrate the absence of errors. Participants were instructed to comprehend the program as well as possible, and they could interact with Jeliot as they found necessary. The duration of a session was not limited and participants were told about it.

The first target program was a factorial computation that was used as a warm-up and the resulting data were discarded. The order of the two actual comprehension tasks was randomized so that half of the participants started with the recursive binary search and other half with naïve string matching.

4 Data Analysis Methodology

In the present experiment, the participants were not limited in the ways they used and interacted with the visualization tool nor in the time they needed to spend on the comprehension. Some of the participants performed only one animation run, while others, in order to comprehend the program as best as they could, needed the help of the dynamic animation more. Since we are interested in commonalities during the comprehension process and to allow for a fair comparison, the whole stream of eye-movement data for a single participant was segmented into the sections where the animation was on and off. The results are therefore in a form of sparse matrix (see a hypothetical example in Table 1 below), depending on the number of animation runs (the columns marked Run 1 ... 4) each participant performed. The columns then contain a measurement obtained during first, second, and following animations of a program.

Let's consider now the situation in which a participant C (Table 1, for fixation count ration between two areas of interest) used the visualization tool to animate the program execution four times, while participant D only two times. Not only the analysis of variance would suffer from a low degree of freedom as the number of animation runs decreases, but potentially, such an approach would align two measures (e.g. in column Run 2) that reflect different cognitive tasks. While participant C was, supposedly, still attempting to create a hypothesis of the visualized problem, the participant D was most probably confirming the hypothesis she created during the Run 1. Therefore, a research question whether the ratio of fixations between two areas is different during Run 1 and Run 2, so that it reflects a shift in the relative importance of the two areas, would be not only ill-defined but also hard to answer using traditional ANOVA.

Participant	Run 1	Run 2	Run 3	Run 4
A	1.162			
B	1.170	1.288	1.978	
C	1.234	1.562	1.619	1.432
D	2.106	2.670		
E	1.953	1.321	2.342	

Table 1: Example of sparse data: ratio of fixation counts on two areas of interest

Typically, repeated measures designs are analyzed using analysis of variance (ANOVA) procedures. However, in the situations when the result matrix is sparse, of few cases, and matching up of corresponding trials between participants is not explicit, as found also in the present experiment, the ANOVA is ruled out. We suggest a possible solution to this problem by reframing the data set as binomial trials and using nonparametric statistical methods to analyze the outcome. This approach allows for answering the research questions related to trends over time and solves the problem with sparse data.

The original data set is considered as whether the scores on successive trials in respect to all previous trials of a participant were higher (an observed increase) or lower (a decrease). While reframing the original data into binomial trials, the research hypothesis needs to be also restated. The claims of null-hypothesis and alternative hypotheses are then stated as:

- H0: *"There is no trend in the trials and same number of increases and decreases will occur."*
- HA: *"There is a trend in the trials and the number of increases and decreases will differ significantly."*

Table 2 shows the result of applying such a transformation from original data contained in Table 1. For example of the Participant E, the first "0" represents no increase in the measurement between Run 1 and Run2, while the consecutive "1" signals an increase in the measurement between Run 1 and Run 3. As for participant A who performed only one Run, no trend it the measurement could be analyzed.

Having the original problem reframed into binomial distribution, the statistical analysis of the newly formed data consists of: 1) counting the total number of trials *N*, 2) counting the cases in which

Participant	1-2	1-3	1-4	2-3	2-4	3-4
A	-	-	-	-	-	-
B	1	1	-	1	-	-
C	1	1	1	1	0	0
D	1	-	-	-	-	-
E	0	1	-	1	-	-

Table 2: Data from Table 1 in binomial form.

the trend was increasing *x*, 3) setting the probability of null hypothesis to $p = .5$ since the variable is binary, 4) computing the cumulative probability by using binomial formula (1) for all i = x..N, and 5) finding the mean value and standard deviation of this distribution.

$$P(X \geq x) = \sum_{i=x}^{N} P(X = i) = \frac{N!}{i!(N-i)!}(p)^i(1-p)^{(N-i)} \qquad (1)$$

Following the example given in Table 2, N = 13, x = 10, p = .5, then $P(X \geq 10) = .0461$, with mean value of 6.5 and standard deviation of 1.8. That means, that the probability of getting greater than or equal to 10 increases out of 13 trials, given the probability of having no trend is .5 (null hypothesis), is .0461. Shall the null hypothesis hold (no trend) the mean number of increases would have to be 6.5 (SD = 1.8). However, as the number of successive increases was 10, we can conclude that we have found an increasing trend with proportion of 10 increases to 3 decreases (76.9%), with an exact probability of getting the same or greater number of increases, assuming the null hypothesis was p = .5, of .0461. Therefore, the null-hypothesis is rejected.

5 Results

In this paper, we study coordination of multiple program representations during program comprehension. We report results related to the time spent on animating the program execution, rather than reporting mean values for a whole comprehension session. We do so because the task was complex and participants were allowed to select their own strategy. Some of the more experienced participants first did not use the animation but instead studied the source code only. Therefore, they did not perform any coordination of multiple representations; these were available during animation only. To allow for a fair comparison, we split the whole stream of data into distinct phases of code-reading and animation and analyze the data during animation.

A detailed analysis of comprehension summaries and interaction patterns is reported and discussed in [Bednarik et al. 2006]. The effects of experience levels on gaze patterns during program comprehension have been reported in [Bednarik et al. 2005].

We analyzed several measures related to the gaze behavior. First, a ratio between the code and visualization in terms of total fixation times was measured. This ratio shall reveal the importance of the two representations during the comprehension process. Next, the number of attention switches per minute between any of the areas of interest shall reflect the dynamics of attention allocation. Finally, we analyzed fixation durations in overall, on the code representation of the program and on graphical visualizations. Fixation durations are believed to reflect relative importance and complexity of information during problem-solving[Just and Carpenter 1976]. The longer the durations are, the more mental efforts a participant has to exert.

Although being an important representation during program comprehension, the output area of Jeliot interface was excluded from the analysis due to the low amount of output of the programs. Since the purpose of this experiment was to explore *how the roles of different representations of a program develop* during program comprehension, the control area of Jeliot interface was also excluded, as it does not contain any information about the program being visualized.

Because of the experimental design, commonly used ANOVA could not be applied for some part of the results. The results were therefore analyzed in part using the method described in section 4, ANOVA and using t-tests for pre-planned comparisons.

5.1 Interaction with the Tool

In overall, the grand mean number of program animation replays was 2.21 (SD = 0.98), with a minimum of no animation, and a maximum of 5. On average, there were 2.06 (SD = 1.2) and 2.25 (SD = 1.2) animations of binary-search program and naïve string-matching program initiated, respectively. There was no difference in the number of animations between the two target programs, pairwise t(15) = .64, *ns*. For the recursive binary-search program, five participants animated the program only once and one did not animate at all, while for the na*i*ve string matching, six participants replayed the animation only once. Therefore, the analysis of eye-movement patterns in the following subsections is based on data of remaining ten participants that could be considered as binomial trials.

Two groups were formed post-hoc from all participants, depending whether a participant animated the programs on average more or less than two times. Table 3 details the distribution of expertise in the resulting groups and resulting statistical differences.

Group	#	Replays	Progr.	Java	Other	Jeliot	Prof.
≤ 2	9	1.25 (0.38)	73.5 (69.92)	16.9 (16.4)	29.6 (38.1)	2	2
> 2	7	3.21 (0.49)	18.0 (19.4)	7.3 (4.1)	5.1 (4.9)	3	0
p		$< .001$	.014	.044	.034	ns	ns

Table 3: Characteristics and differences (means, standard deviations in parentheses) of the two groups (≤ 2 animations, > 2 animations) in months (Programming, Java, Other experience), analyzed by one-tailed t-test, and number of participants (Jeliot experience, Professional experience), by χ^2 test.

5.2 Animation Replays as Binomial Trials

For the purposes of the eye-movement analysis using binomial distribution, the replays of the animations were transformed into binomial trials. Thus, during the animations of binary-search program, 28 trials were performed. The resulting binomial distribution for null hypothesis (p = 0.5, i.e. no trend with same number of increases and decreases) have a mean of 14 with a standard deviation of 2.65. During the animations of naïve string matching program 37 trials were performed, that formed a binomial distribution with a mean of 18.5 and a standard deviation equal to 3.04.

5.3 Fixation Time

To estimate the importance of the available representations, we computed the total fixation time as a sum of all fixation durations on the two main areas of interest during each of the animation runs. Then, for each of the animations a participant initiated, we calculated *the ratio of the fixation time spent on code to the time spent fixating the graphical representation*. For those participants who animated at least twice, these ratios were then transformed into binomial trials as described in Section 4.

The mean ratio of total fixation times on code/visualization of remaining participants that animated the programs only once was 0.86 (SD = 0.5) and 0.65 (SD = 0.32) for recursive binary search and for naïve string-matching, respectively. There was no statistical difference in the mean ratios between the two programs of this group, independent-sample t(8) = 0.77, *ns*.

Twenty-two increasing trials and six decreases were found during the replays of recursive binary search. Therefore, the ratio of fixation time spent on code/visualization had an increasing tendency, p = *.0019*. Twenty-one increases and 16 decreases were found during the replays of naïve string matching animation. Thus, the ratio of fixation time spent on code/visualization had a slightly increasing trend, p = *.26*, also in this program.

5.4 Attention Switching Behavior

The dynamics of attention allocation was measured as *a number of switches per minute* between any of the areas of interest during the animation. We define a switch as any change of the focus of visual attention between any of the pre-defined areas of interest.

Those participants that animated the program execution only once, performed 40.6 (SD = 22.7) and 48.8 (SD = 9.8) attention switches per minute, while comprehending binary-search and string-matching programs, respectively. There was no difference in the number of switches per minute between the programs, independent-sample t(8) = 0.74, *ns*, of this group.

Attention switching behavior of the group that animated program execution more than once can be characterized as 10 increases in the number of switches per minute and 18 decreases, p = .96, in binary-search program, and as 11 increases and 26 decreases, p = .99, in naïve-string matching. Therefore, in both of the target comprehension tasks, the number of attention switches per minute performed by the participants shown a strongly decreasing trend.

5.5 Fixation Duration

The interface of the program visualization tool consisted of two main areas of interest, the code and the visualization panel (Theater). Besides for the code as a main representation, we measured fixation durations also for the four discrete areas of the Theater that show certain aspects of the program execution (described in section 3.2).

Mean fixation durations of the group that animated the programs only once (Table 4) were analyzed using ANOVA (program(2) x area (5)); missing values were replaced by the means of the group for that particular area and program. No effect of program on fixation duration was found, F(1,8) = 1.28, *ns*, together with no interaction with area, F(4,32) = .51, *ns*. However, there was a major effect of area on fixation duration, F(4,32) = 3.22, p = *.025*. In both programs, the longest durations were measured on the area containing visualization of expression evaluation, while the shortest durations fell on the area displaying constants.

Fixation durations of the group that animated the programs more than once were analyzed using the binomial trials (Table 5). Several

Program	Code	Methods	Expressions	Instances	Constants	Overall
binary-search	301.7	285.5	387.3	293.9	249.5	312.8
	(137.4)	(80.6)	(159.6)	(189.5)	(53.4)	(120.1)
string-matching	273.7	249.2	317.9	276.6	214.2	284.3
	(26.7)	(16.7)	(93.3)	(13.5)	(32.2)	(38.1)

Table 4: Mean fixation durations and standard deviations (in parentheses) over the main areas of interest and in overall for the group that animated one time.

general trends can be observed, regardless of the program. Most importantly, the steepest increasing trend is related the fixation durations on the code area. On the other hand, the fixations that fell on the area containing instances of objects and arrays were decreasing with following animation runs. No trends were observed in areas visualizing expression evaluations. Altogether, participant's fixation durations on the areas of the Theater were decreasing or remained about same while those on the code area, on the other hand, were showing an increasing trend. Finally, the overall fixation duration in both of the programs seemed to stay same during consecutive animations.

6 Discussion

Unlike many previous studies that involved some variation of think-aloud protocol to trace the subjects thought processes, we have conducted an exploratory study of program comprehension on unfamiliar code using eye-movement tracking. Our aims were to introduce eye-tracking as a valid source of data about behavioral aspects of programming and to develop an approach to analyze this data. Although the study was designed as a repeated measures, traditional statistical methods for analyzing the data could not be applied. Therefore, we reframed the results and considered the data as binomial trials.

6.1 Eye-movement Data as Binomial Trials

In comparison with other more traditional approaches to statistical analysis of eye-movement data of repeated measures designs, the methodology introduced in this paper provides researchers with a possibility to investigate trends over time. Using average values of collapsed measurements and some methods of analysis of variance might not provide enough information about how strategies of experimental participants develop during problem solving and learning.

However, also practical considerations have to be taken into account when applying this approach. First, the valuable data from those participants that performed only one trial are not considered and this dissipates the efforts spent on recruiting the participants. In the present experiment, fortunately, these participants belonged to the group whose behavior was different from the rest of sample population. Second, the reframing of the original data represents an additional step in the analysis. However, the computations do not require any special techniques and, in our opinion, are justified by the possibility to interpret eye-movement data as trends.

6.2 Limitations of the Data Analysis

The proposed methodology of analyzing sparse data that originated from unequal number of trials in a repeated measure design seems to be practical when the research questions do not concern the exact values of the measures, but rather investigate trends in a time. Although the proposed approach provides many advantages in these situations, several limitations can be identified. Most importantly, by converting the data sets to binomial trials the nominal values of original data are not part of the results any more. This makes the results hard to compare to other studies that used parametric methods to analyze their results and reported means of their measures. However, as the resulting probabilities are reported and exact, these can be used for a comparison. Second problem, that will be addressed in future studies, is that the binomial distribution requires all trials to be independent. While this condition holds for between-participant trials, within-participants trials can hardly be considered independent.

6.3 Comprehension process analysis

With the exception of one experienced participant during one program comprehension session, all participants animated the program execution at least once. Mean number of animation runs turned out to be a good indicator of previous programming experience. Those, who on average run the animation at most twice belonged to a more experienced group, especially in terms of general experience in programming. This finding is not surprising and can be explained by at least two reasons. First, although we provided all participants with a short introduction to Jeliot and we tried to equalize the previous experience with the tool by having a warm-up session, most of the experts in our study were not previously familiar with the tool and with the visualization. As it has been argued, to benefit from program visualization, users have to be explicitly taught to use it [Ben-Ari 2001]. Therefore, experts' low number of animation runs could be explained by their unfamiliarity with the tool. However, the previous experience with Jeliot was balanced between the groups and still those participants in the less experienced group who have animated the program several times were lacking the previous training with the tool.

Second reason of the low number of animations of experts can stem from a difference in the strategies to comprehend the program using the visualization tool [Bednarik et al. 2006]. According to Brooks [1983], once a hypothesis of the program is formed, the programmer tries to verify it against the program text. In our experiment, more experienced participants read the code first, created a model and hypotheses, and then confirmed their hypotheses by (usually) only one run of the animation.

On the other hand, less experienced programmers did not read the code at the beginning, but instead animated the program several times and let the tool to visually explain the execution. They engaged in doing so, until they have collected enough information to form a hypothesis and then, during the last run, coordinated the representations in the way that favored code more than at the beginning, to confirm their hypotheses.

The results obtained by capturing and analyzing the eye-movement data can be seen as supporting the discussion above. At the beginning of the comprehension, less experienced participants targeted

Program/Trials		Code	Methods	Expressions	Instances	Constants	Overall
binary-search/28	Increases	17	9	11	4	12	12
	p	0.17	0.98	0.91	1.0	0.83	0.83
string-matching/37	Increases	21	18	19	17	20	18
	p	0.25	0.63	0.50	0.74	0.37	0.63

Table 5: Fixation durations over the main areas of interest and in overall, as binomial trials with associated probabilities.

their attention toward the visualization of the target program, as indicated by the total fixation time. However, with increasing iterations, the less experienced participants focused more on the code representation.

Considering the dynamics of the coordination strategies, participants gradually decreased the attention switches between the code and the visualization, and concentrated on the code representation. In other words, the attention allocation and related eye-movements reflect what information and what representation is relevant to the programmers during the comprehension task. It turned out that at the early phases of comprehension, visualization provides more important information and plays more important role than at the later stages.

More experienced programmers exhibited different behavior. During their single animation run, they spent more time focusing on visualization than on the code. Based on the mean fixation durations over the discrete areas of visualization, we believe that the more experienced programmers faced highest difficultness with the expression evaluations, while following the visualization of constants did not introduce similar depth of cognitive processing.

As discussed before, expert programmers, in this experiment, studied the code of the program first before they initiated the visualization. The visualization then provided them with an additional information, most probably needed to confirm and fine-tune their previously established mental model and hypotheses.

6.4 Limitations of the Experiment

Some limitations can be identified considering the experimental part of this study. The number of participants was relatively low and further decreased by technical problems with eye-movement tracking. This issue will be addressed in future studies to ensure higher validity of data and drawn conclusions.

Another limitation is related to the generalizability of the findings, concerning the gaze behavior during comprehension. Although the current program visualization systems that are in use share many external features, such as design of interface (e.g. the horizontal split of main area, code on left, visualization on right), their inner mechanism and interaction with the tools might differ. For example, the visualization of Jeliot 3 is based on representing a virtual machine on which a program is executed, while visualizations of PlanAni [Sajaniemi and Kuittinen 2003] are based on a cognitive concept. In addition, some of the previous experiments in program comprehension and debugging with multiple representations involved only precomputed static or semi-dynamic visualizations [Romero et al. 2002]. Therefore, the eventual behavior recorded during comprehension with other tools might be different for any of the involved experience groups.

7 Conclusions and Further Work

Eye-movement based analysis can contribute to our understanding of cognitive processes involved in program comprehension, debugging, and visualization. Unlike verbal reports used in numerous previous studies of program comprehension, eye-movement-based analysis does not require any training of experimental participants to verbalize their thoughts neither interferes with their mental processing.

This paper demonstrated, that in incorporation with other experimental protocols, eye-tracking data can reveal important information about behavior of computer programmers, that can be hard to access using only a single methodology. Reframing of original repeated-measures eye-movement data as the sets of binomial trials allowed to characterize the development of program comprehension behavior in terms of representation use and coordination.

However, as the applications of eye-movement tracking to program comprehension are rare up to date, more studies have to be conducted and the methodological framework needs to be further developed. The framework shall provide researchers with a tool to capture, analyze, and explain the cognitive processing during program comprehension using eye-movement data.

8 Acknowledgement

We would like to thank all participants for taking part in the experiments. Help of Niko Myller before, during, and after the experiment has been greatly appreciated, as well as that of Justus Randolph whose idea initiated this work. Several anonymous reviewers provided valuable comments on an earlier version of the manuscript.

References

Bednarik, R., and Tukiainen, M. 2004. Visual attention tracking during program debugging. In *Proceedings of The Third Nordic Conference on Human-Computer Interaction (NordiCHI'04)*, ACM Press, New York, NY, USA, 331–334.

Bednarik, R., and Tukiainen, M. 2005. Effects of display blurring on the behavior of novices and experts during program debugging. In *Proceedings of (CHI'05)*, ACM Press, Portland, OR, USA, Extended abstracts of the ACM Conference on Human Factors in Computing Systems, 1204–1207.

Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M. 2005. Effects of experience on gaze behaviour during program animation. In *Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05)*, 49–61.

Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M. 2006. Analyzing individual differences in program comprehension with rich data. *To appear in Technology, Instruction, Cognition and Learning*.

BEN-ARI, M. 2001. Program visualization in theory and practice. *Informatik/Informatique 2*, 8–11.

BLACKWELL, A. F. 2002. First steps in programming: A rationale for attention investment models. In *HCC '02: Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)*, IEEE Computer Society, Washington, DC, USA, 2.

BRANCH, J. L. 2000. Investigating the information-seeking processes of adolescents: The value of using think alouds and think afters. *Library and Information Science Research 22*, 4, 371–392.

BROOKS, R. 1983. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies 18*, 543–554.

BURKHARDT, J., DÉTIENNE, F., AND WIEDENBECK, S. 2002. Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering 7*, 2, 115–156.

CROSBY, M. E., AND STELOVSKY, J. 1990. How do we read algorithms? A case study. *IEEE Computer 23*, 1, 24–35.

ERICSSON, K. A., AND SIMON, H. A. 1984. *Protocol analysis: Verbal reports as data*. Braford Books/MIT Press, Cambridge, MA.

GOOD, J., AND BRNA, P. 2004. Program comprehension and authentic measurement: : a scheme for analysing descriptions of programs. *International Journal of Human Computer Studies 61*, 2, 169–185.

JANSEN, A. R., BLACKWELL, A. F., AND MARRIOTT, K. 2003. A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, and Computers 35*, 1, 57–69.

JUST, M. A., AND CARPENTER, P. A. 1976. Eye fixations and cognitive processes. *Cognitive Psychology 8*, 441–480.

KO, A. J., AND UTTL, B. 2003. Individual differences in program comprehension strategies in unfamiliar programming systems. In *IWPC '03: Proceedings of the 11th IEEE International Workshop on Program Comprehension*, IEEE Computer Society, Washington, DC, USA, 175.

LETOVSKY, S. 1986. Cognitive processes in program comprehension. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, Ablex Publishing Corp., Norwood, NJ, USA, 58–79.

LITTMAN, D. C., PINTO, J., LETOVSKY, S., AND SOLOWAY, E. 1986. Mental models and software maintenance. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, Ablex Publishing Corp., Norwood, NJ, USA, 80–98.

MORENO, A., MYLLER, N., SUTINEN, E., AND BEN-ARI, M. 2004. Visualizing programs with Jeliot 3. In *Proceedings of the Working Conference on Advance Visual Interfaces (AVI 2004)*, ACM, 373–376.

NEVALAINEN, S., AND SAJANIEMI, J. 2005. 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)*, 77–91.

NIELSEN, J., CLEMMENSEN, T., AND YSSING, C. 2002. Getting access to what goes on in peoples heads? reflections on the think-aloud technique. In *Proceedings of The Second Nordic Conference on Human-Computer Interaction (NordiCHI'02)*, ACM Press, New York, NY, USA, 101–110.

PETRE, M. 1995. Why looking isn't always seeing: readership skills and graphical programming. *Communications of ACM 38*, 6, 33–44.

RAYNER, K. 1998. Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin 124*, 372–422.

ROMERO, P., LUTZ, R., COX, R., AND DU BOULAY, B. 2002. 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)*, IEEE Computer Society, Washington, DC, USA, 207.

SAJANIEMI, J., AND KUITTINEN, M. 2003. Program animation based on the roles of variables. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, ACM Press, New York, NY, USA, 7–16.

SOLOWAY, E., LAMPERT, R., LETOVSKY, S., LITTMAN, D., AND PINTO, J. 1988. Designing documentation to compensate for delocalized plans. *Communications of ACM 31*, 11, 1259–1267.

VAN DEN HAAK, M., JONG, M. D., AND SCHELLENS, P. J. 2003. Retrospective vs. concurrent think-aloud protocols: testing the usability of an online library catalogue. *Behaviour and Information Technology 22*, 5, 339–351.

VON MAYRHAUSER, A., AND VANS, A. M. 1996. Identification of dynamic comprehension processes during large scale maintenance. *IEEE Transactions on Software Engineering 22*, 6, 424–437.

P6.

6

Bednarik, R., Tukiainen, M.: **Analysing and Interpreting Quantitative Eye-Tracking Data in Studies of Programming: Phases of Debugging with Multiple Representations.** In Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group (PPIG'07), Joensuu, Finland, July 2-6, 2007, pp. 158-172

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].

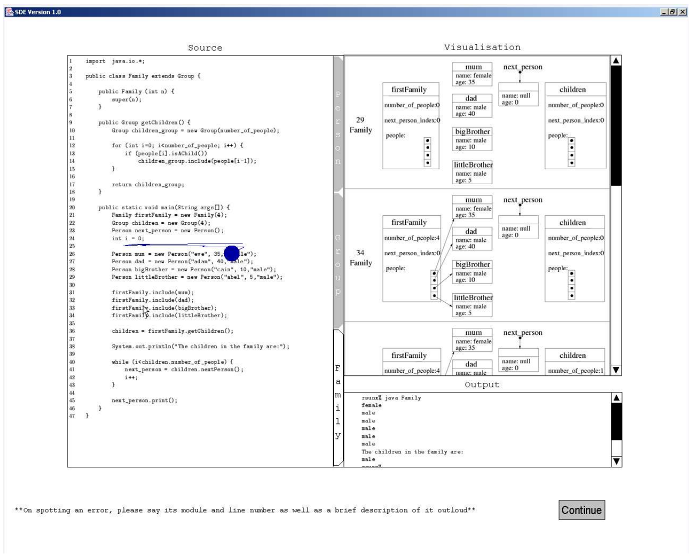
Method Figure 1 presents a screenshot of the IDE during the non-restrictive condition. The environment presented the source code of a Java program (left pane in Figure 1) that contained four non-syntactic errors. Participants – after reading specifications of the desired behavior of the program – were 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,"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

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.

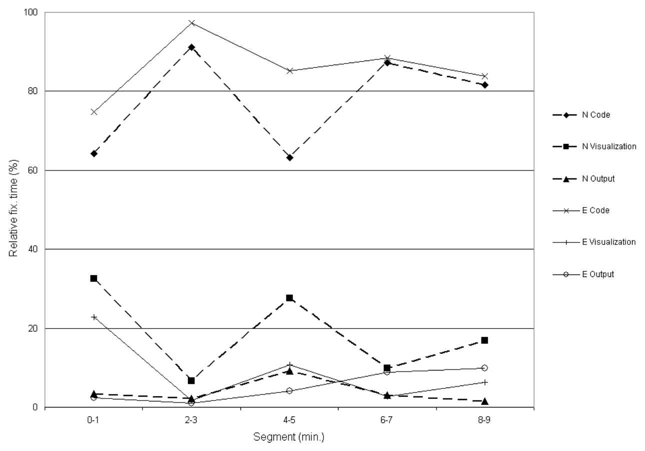


Fig. 2. Plot of proportional fixation times during five phases of debugging.

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.

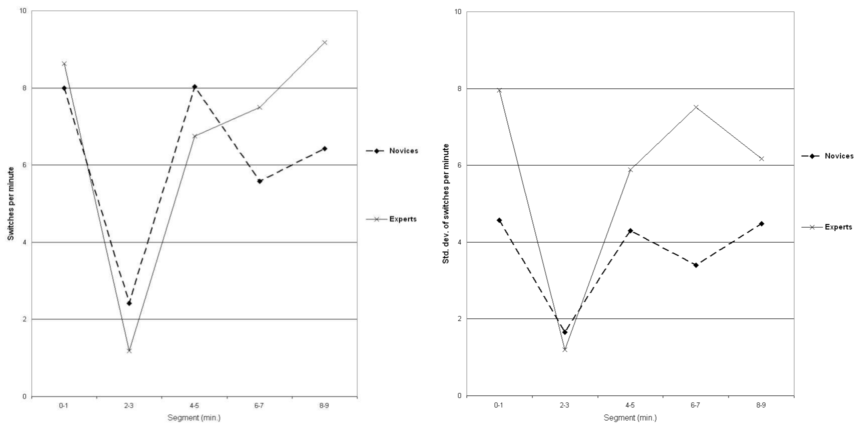


Fig. 3. Plots of switching behavior ((a) mean, (b) standard deviation) during five phases of debugging.

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

Dissertations at the Department of Computer Science and Statistics

Julkaisija	Joensuun yliopisto Tietojenkäsittelytieteen ja tilastotieteen laitos
Publisher	University of Joensuu Department of Computer Science and Statistics
Sarjan toimittaja Series Editor	Erkki Sutinen
Vaihdot	Joensuun yliopiston kirjasto/Vaihdot PL 107, 80101 Joensuu Puh. 013-251 2677, fax 013-251 2691 email: vaihdot@joensuu.fi
Exchanges	Joensuu University Library/Exchanges P.O. Box 107, FI-80101 Joensuu, FINLAND Tel. +358-13-251 2677, fax +358-13-251 2691 email: vaihdot@joensuu.fi
Myynti	Joensuun yliopiston kirjasto/Julkaisujen myynti PL 107, 80101 Joensuu Puh. 013-251 4509, fax 013-251 2691 email: joepub@joensuu.fi
Sales	Joensuu University Library/Sales of publications P.O. Box 107, FI-80101 Joensuu, FINLAND Tel. +358-13-251 4509, fax +358-13-251 2691 email: joepub@joensuu.fi