Program Visualization: Comparing Eye-Tracking Patterns with Comprehension Summaries and Performance

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

University of Joensuu, PO BOX 111, FI-80101, FINLAND,
{bednarik,nmyller,sutinen,mtuki}@cs.joensuu.fi,
http://www.joensuu.fi/tkt

Abstract. We present preliminary results of an experiment in computer program comprehension that was conducted to find out whether visual strategies can characterize low- and high-comprehenders. In addition, we investigated whether the type and quality of externalized mental models can be associated with the visual strategies.
Participants of various levels of experience used a program visualization tool, Jeliot, to comprehend short Java programs, while their eye-movements were recorded. Comprehension summaries were evaluated for correctness as a measure of performance and also analyzed using Good's information-types scheme. Times spent on viewing certain structures of the program visualization were analyzed and correlated with the information types found in comprehension summaries. Depending on comprehension performance and target program, some information types were found to be correlated with eye-data patterns.
Comprehension performance did not significantly correlate with information types. When the visual strategies of low-comprehenders were similar to those of high-comprehenders, the comprehension outcome of the low-comprehenders was poor. When the strategies diverged, the mental models of low-comprehenders tend to match those of high-comprehenders. Based on the results, we propose that eye-tracking can help to partially predict the mental model that is built during comprehension. We discuss limitations and future directions of this research.

1 Introduction

Program comprehension, the ability to understand programs, is often recognized as central to programming and software maintenance in general. Researchers examining cognitive processing have used several techniques, such as think-aloud protocols, observational studies, eye tracking and other, to get insight into the behavior and strategies the participants exhibit during reading or problem-solving tasks. Previous research in program comprehension have used many of these methods to capture and investigate cognitive processes involved in programming [1–4]. However, visual attention tracking methods have been employed relatively rarely.

When the reasoning is related to or even dependent on visual stimuli, eye-tracking systems have shown to be useful in revealing the patterns of visual attention during the task. Programming tasks such as design, debugging or comprehension are such situations, since normally programmers work with a graphical interface that provides them with several representations of a program. While most of the previous studies have utilized think-aloud protocols and artificial environments, visual attention patterns during programming have not been studied widely. Little has been done to analyze and explain the visual strategies of programmers and relate them to the cognitive models formed during the programming tasks. Therefore, one of the motivations of this study is to investigate the usability and also limitations of gaze tracking for studies of programming. We also wish to relate the gaze-data to and validate them with other sources of empirical and behavioral data.

Our longterm goal is to characterize the visual strategies and related mental models of programmers of different skill levels and performance in program comprehension. As a step in our ongoing efforts, we present an exploratory empirical experiment in which participants tried to comprehend Java programs using a program visualization tool. To capture the visual attention patterns during comprehension, we recorded the eye-movement data. Our study aims to address questions such as: *What kinds of visual strategies lead to successful and unsuccessful comprehension? How is the quality of constructed mental models related to the visual attention patterns during comprehension? What structures do programmers visually attend in order to comprehend a program successfully?* Answers to these questions help us to better understand not only what shall be visualized, but also how program visualization is used and should be used [5]. Furthermore, this research contributes to systems that can support programmers and students based on their eye-data during the programming tasks.

2 Related Work

What cognitive processes are involved during programming tasks? One way to study cognitive skills is through examining differences in the performance of novices and experts [6]. Differences between novices and experts in comprehension and debugging tasks have been of great interest in previous research. Gugerty and Olson [7] found that during debugging, not only novices were much slower in discovering the bug, but they also introduced new bugs. Experts tend to spend more time on planning and evaluation, and their mental models are rich to support mental simulations of the programs [8]. Experts also think about programs in terms of higher-level abstractions [3], while novices use elements on the (surface) level of the programming language [9].

Eye-movement tracking has been successfully applied in many domains, including cognitive processing [10], reading [11], usability [12], or as a medium for direct interaction with interfaces [13]. In the domain of empirical studies of programming, instead, investigations of cognitive processes involved in programming have been mostly based on verbal-protocols [4], a well established - and probably the most popular - method, used to capture and analyze the thought-processing. Despite its potentials, eye-movement tracking has not been widely applied in the domain and the applications of the eye-movement tracking to study the behavioral aspects of programming are still rare.

The advantages of eye-tracking to study human behavior are, however, numerous. Modern eye-trackers are highly unobtrusive and no additional effort is required from participants to be tracked. Capitalizing on these advantages, Stein and Brennan [14] used a head-mounted eye-tracker to record the point of gaze of professional programmers debugging. These recordings were then replayed for a half of other participants searching for the bugs. It was found that those who viewed the gaze path of professionals found the bugs more quickly than those who did not see it. Torii et al. [15] made use of the gaze from a wearable eye-tracker as one of the sources of behavioral data to monitor users during software development.

Crosby and Stelovsky [16] studied gaze patterns of participants reading source code. They found that the visual strategies vary and depend on individual preferences. In addition, beacons [17], the typical structures found in the source code, were found to play an important role in visual strategies. Using an artificial environment and multiple static representations, Romero et al. [18] studied visual attention during debugging. Good debugging performance was found to be linked with balanced switching between the different representations.

In a previous analysis of the present experiment, we concentrated on the *effects of previous experience* on the interaction and gaze patterns during comprehension [19]. In the experiment, participants with various levels of programming experience used a program visualization tool, Jeliot [20], to comprehend short Java programs. We have found a significant effect of previous experience 1) on the general comprehension strategies outside the animation, 2) on the fixation durations over different areas of interest, and 3) on the interaction patterns with the visualization tool. More experienced participants spent more time on reading the code and generating hypotheses to finally validate them against the visualization. Novice programmers, on the other hand, viewed the visualizations in order to generate hypotheses without studying the code carefully first. Only then they tried to comprehend the programs from the source code. However, the quality of the resulting mental models did not significantly differ and gaze patterns during the visualization were similar to a great extent. Novice participants exhibited a higher mean fixation duration that can be related to the depth and complexity of the required mental processing.

In a study similar in some aspects to the present one, Nevalainen and Sajaniemi [21] investigated the effects of a programming environment, Turbo Pascal, and a program visualization system, PlanAni, on visual strategies and mental models constructed during program comprehension. Eye-movement data were analyzed in terms of the proportion of fixation times spent on code and visualization. While the authors found some effects of the tools on the visual strategies, no statistically significant effects on the constructed mental models were found. Some correlations between looking into visualization of variables and information types were found. However, a great number of correlation tests were performed in the study. It is then probable that many of the significant findings might not indicate real dependencies, but be artifacts of the type I error because the significance level was not adjusted.

Bednarik et al. [19] claim that complex and lengthy processes such as program comprehension cannot be effectively described using a single long-term eye-tracking measure. In addition, the visualization of program execution is dynamic and often involves several semantically distinct structures and entities that appear on the display during limited period. For instance, a dedicated area shows the type and actual content of variables, while another one concurrently illustrates changes in the control flow. Thus, we study how the visual attention is allocated on those elements during the time they are active. In the present study, we therefore concentrate only on the times when the animation of a program was available and we decompose the visualization interface into several semantically distinctive areas.

Programmers acquire a mental model of a program as an outcome of the comprehension process. Good and Brna [22] developed an analysis scheme based on the information types and object descriptions found in comprehension summaries. The scheme allows the evaluation of the quality of mental models in terms of proportions of different information types and in terms of their level of abstraction. The information types are the statements related to control-flow, data-flow, function or operation of the program. Object descriptions (i.e. how the variables and objects are explained in the summary) are classified based on the references they make to program, domain or real-world specific terms. A more detailed description of the comprehension summary analysis can be found in [22].

In this report, we focus on particularly successful comprehenders and contrast their visual strategies with those of low-comprehenders. We make use of Good's scheme and analyze the gaze-patterns of each of the groups and make an attempt to correlate the mental models with the visual strategies. In addition, we compare the comprehension performance to the information types, to investigate whether quality and performance correlate.

3 Experiment

The purpose of this exploratory investigation was to discover if high and low comprehension outcomes are results of particular visual strategies of programmers using a program visualization tool. Furthermore, we investigated whether information types found in the program comprehension summaries correlate with these strategies. Knowledge of such patterns could have direct implications to the design of future programming environments that could use the gaze-tracking in real time.

3.1 Method

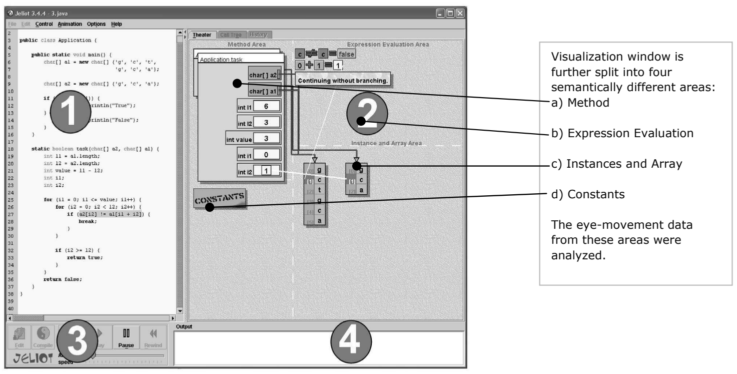
Dependent variables were the information types that were found in program comprehension summaries, comprehension summaries' quality measures, and the proportions of fixation times spent on each area of interest. The information types reflect the quality of mental models. The proportional fixation time is a measure of participant's interest on an area [12] and therefore it reflects the importance of the information contained in the area. Only the gaze data during the program animation were used in this analysis because that was the only time when all the representations were available concurrently. Thus, the selection of the attended representation would make a difference in understanding the program. The data were analyzed using correlations, ANOVA and planned comparisons based on t-test.

3.2 Participants

A total of sixteen (13 male, 3 female) participants were recruited from high-school students attending a university level programming course, and undergraduate and graduate computer science students from local university. The experimental group can be characterized as follows with mean values (standard deviations in parentheses): age 23.25 (7.67), experience with programming in months 49.13 (54.05), experience with Java in months 13.06 (12.75), experience with other programming languages 19.31 (29.25) months, two participants had a previous industrial experience. All participants had normal or corrected-to normal vision, according to their own report.

3.3 Materials and Apparatus

Three short Java programs, a factorial computation, a recursive binary search program, and a naïve string matching program were presented to the participants. 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 in order to avoid a possible recognition of a program based on these surface features and motivate the participants to try to understand the programs.



```
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 ([illegible])) {
            [illegible]println("True");
        } [illegible]
            [illegible]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;
    }
}
```

Visualization window is further split into four semantically different areas:
a) Method

b) Expression Evaluation

c) Instances and Array

d) Constants

The eye-movement data from these areas were analyzed.

Fig. 1. Interface of the program visualization tool used in the experiment.

To visualize the target Java programs, Jeliot 3 [20], a program visualization tool, was used. The user interface of Jeliot 3 (Figure 1) consists of four separate areas: the Code (1) is on the top left, the visualization is shown in the top right area (2), the Control panel (3) with VCR-like buttons to control the animation is on the bottom left, and the Output (4) of the program is displayed in the bottom right panel.

Moreover, the visualization area is further split into four discrete sections that detail a) the method frames and local variables, b) expression evaluation, c) constants and static fields, and d) instantiated objects and arrays. These four areas were in the focus of the present study. The *Method area* displays the currently executed method and types and values of local variables. When an expression is being evaluated during execution, the *Expression Evaluation area* displays the process step-by-step. In addition, information about method calls, return values and some explanations about the control flow of the program are presented in this area. The *Instances and Arrays area* shows the visualization of arrays and instances of objects, their fields and content. Finally, in the *Constants and Static Fields area* all the literal values are introduced and static fields are visualized. The areas are not separate in the sense that there are several operations that can transfer information from one area to another. For instance, when a result of an expression evaluation is assigned into a local variable, the resulting value is moved from the expression evaluation area to its proper location in the method area. Thus, these transitions contain also semantical meaning related to the programs' execution.

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. 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. In addition they were told that after the comprehension they will be asked to write a summary of a program. They could interact with the program visualization tool 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 target 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

Previous reports of this experiment [19] concentrated on distinguishing visual attention patterns of participants according to their prior programming knowledge and experience. In this report, we focus on particular successful comprehenders and contrast their visual strategies with those of low-comprehenders.

Comprehension summaries were evaluated based on three elements. A point was given to those that contained a correct description of the function (*what*) of a program. Another point was given if the procedure of the program (*how*) was described correctly. If description contained a full description of the program and its execution in the current case, a point was given. Thus, a comprehension summary could be given a maximum of three points.

4.1 Comprehension performance vs. quality of mental models

Participants were post-hoc divided into two groups based on their performance in comprehension: those participants whose comprehension summaries received at least two points in both of the target programs were assigned to a *high comprehenders* group, and other participants were assigned to a *low comprehenders* group. Following the criterion, nine participants were assigned to the high-comprehenders group and seven participants were assigned to the other group.

All thirty two program summaries were analyzed by an experienced rater. To establish the validity of the information-types analysis and of the performance evaluations, eleven randomly selected summaries (approximately one third) were analyzed by two raters. The summaries were chunked based on consensus, resulting in 130 chunks. The pure inter-rater agreements were 83.8% (κ=.814, ASE=.037, *p<.001*) regarding the information types analysis. Considering the comprehension performance analysis, 100% for presence of both function and procedure correct descriptions, and 90.9% (κ=.621, ASE=.335, *p=.026*, one disagreement) for correct full description of the program were achieved [1].

High-comprehenders received on average 2.33 (SD=0.44) points, while the other group achieved an average score of 1.07 (SD=0.62). Besides a significant effect of group F(1,14)=32.34, *p<.001* on points received, a two-way ANOVA (program (2) x group (2)) discovered no effect of program F(1,14)=.445, *p=.52*. A weak interaction between group and program F(1,14)=4.43, *p=.054* was found, with partial η^2 = .24. Thus, according to the performance criteria, the groups significantly different (high-comprehenders performed significantly better) and the performance kept constant between the programs. This finding has been expected and the analysis only served as a confirmation that the post-hoc division yielded different groups in terms of performance.

To evaluate the information types (IT) found in summaries and therefore the quality of the externalization of the constructed models, comprehension summaries were analyzed using Good's scheme [22]. To reduce the complexity of the analysis, we concentrated on two composite ratings that would reflect the level of abstractions found in the models. Similarly as in [23, 21], the *info-high* type was defined as a composite value including the higher-level statements about data, function, action, and so called state-high statements. The *info-low* composite type included statements at a lower-level of abstraction, namely statements about operation, control, and state-low. Other information types provided by Good's scheme were not included in the analysis, however, the proportions are reported in Table 1.

Since both the classification of the performance and the information types analysis were based on the same comprehension summaries, it might be hypothesized that the two measures are correlated. To analyze a possible relationship, we correlated the points awarded with the info-high and info-low proportions. For all of the summaries, comprehension points with info-low and comprehension points with info-high were not significantly correlated: info-low and comprehension points had a small negative correlation (pearson *r* between -.28 and -.16, *ns*) and info-high and comprehension points had a small positive correlation (pearson *r* between .25 and .18, *ns*). It therefore seems that the performance classification and analysis of produced mental models were inde-

[1] Inter-rater agreement is considered reliable enough if $\kappa > .7$. However, there are problems applying κ, when there are few categories and values. Thus, the $\kappa = .621$ of 11 agreements/disagreements can be still considered relatively good.

Table 1. Proportions of information types found in comprehension summaries. SD = Standard deviation.

IT	Target program							
	Binary search				String matching			
	High compr.		Low compr.		High compr.		Low compr.	
	mean	SD	mean	SD	mean	SD	mean	SD
function	7.47	3.64	4.29	6.31	6.50	4.63	4.68	4.97
action	7.60	5.80	5.47	7.16	8.55	9.43	15.85	13.43
operation	8.07	16.04	20.83	20.73	12.56	10.69	20.39	15.76
state-high	7.11	7.62	7.20	6.80	14.93	9.84	6.68	3.53
state-low	13.63	12.31	11.09	15.95	5.80	5.79	13.99	13.02
data	21.54	9.51	9.28	8.59	12.85	9.03	6.77	6.05
control	22.16	10.87	23.32	21.02	12.76	9.85	9.77	6.29
elaborate	9.79	11.33	16.60	13.40	23.29	15.09	11.03	9.27
meta	1.79	3.33	0.00	0.00	0.00	0.00	5.96	10.29
unclear	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
incomplete	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
continuation	0.00	0.00	1.91	3.28	0.00	0.00	0.00	0.00
irrelevant	0.83	2.36	0.00	0.00	2.76	4.41	4.89	5.64
Σ info-high	43.72	15.98	26.24	15.29	42.83	17.83	33.15	15.34
Σ info-low	43.86	17.68	55.25	26.30	31.11	18.32	44.15	19.78

pendent in general and can be used as separate measures. In other words, presence of certain information type (e.g. info-high) in a summary does not indicate correct comprehension, and vice versa.

Table 1 provides a complete overview of the comprehension models and information types found in the summaries, for both of the two program comprehension tasks. Three-way ANOVA (program (2) x group (2) x IT-type (2)) revealed no effects of the type, F(1,13)=1.674, *p=.218* and a mild effect of program F(1,13)=3.456, *p=.086* on IT found in mental models, with a weak interaction between IT-type and performance group F(1,13)=2.598, *p=.131*. Other interactions were not significant; similarly as no effect of group on information type was found F(1,13)=.045, *p=.84*. Planned comparisons revealed a significant difference between the groups on info-high-type, t(13)=2.16, *p=.050* for the binary search, and no difference for the string matching t(13)=1.06, *p=.305*. No statistically significant differences were found in the proportions of info-low type.

It shall be observed that the analysis of the information types is strongly affected by the great variances in the models. Nevertheless, the summaries of low-comprehenders contained more low-level information than high-level information. Although the groups statistically did not differ in the proportion of info-low-type found in their summaries, clearly, mental models of high-comprehenders contained higher proportions of higher level references than the models of low-comprehenders.

4.2 Eye-movement patterns vs. quality of mental models

Correlations of eye-movement data on the main interface areas of the interface with the proportions of information types found in the summaries were analyzed. We analyzed the visual behavior separately for the two target programs, because the animations of the program executions were different and they might have had an effect on the allocation of visual attention. Correlations of the times spent on the two main interface areas are shown in Table 2 and the distribution of the times is shown in Table 3.

Table 2. Correlations of proportional fixation times on the Code and Visualization areas with information types. Levels of significance in parentheses. Correlations approaching p=.1 are highlighted in italics.

Program	Group	IT	Code	Visualiz.
Binary search	High compr.	info-high	-.200 (.635)	.235 (.575)
		info-low	.005 (.990)	-.047 (.913)
	Low compr.	info-high	*.679* *(.094)*	*-.622* *(.136)*
		info-low	-.605 (.150)	.484 (.271)
String matching	High compr.	info-high	-.226 (.559)	.223 (.565)
		info-low	.191 (.622)	-.184 (.636)
	Low compr.	info-high	-.079 (.866)	.005 (.992)
		info-low	.103 (.825)	-.072 (.878)

Recursive binary search program Detailed analysis of the fixation patterns on the sub-areas of the visualization window revealed following: while comprehending the binary search program, none of the proportional fixation times were significantly correlated with any of the information contained in mental models, except for a negative correlation of low-comprehenders' time to view the Expression area and info-high that approached p of .05, (r(7)=-.73, *p=.062*). To analyze the effect of code reading during the animation of this program, the time spent on looking into the source code and time spent on looking into visualization were correlated with information types, as shown in Table 2. Although not significantly, the times the low-comprehenders spent on Code and Visualization were interestingly correlated with the info-high and partly with info-low. In addition, the distributions of the fixation times did not differed between the groups during comprehension of this program.

Table 3. Proportional fixation times on the Visualization areas (Method + Expression + Instances + Constants = Visualization) and on the source code and Visualization [in % of total animation time]. Probabilities are associated with two-tailed t-test, standard deviations are in parentheses.

Program	Group	Method	Expression	Instances	Constants	Code	Visualization
Binary search	High compr.	24.02	21.07	4.73	0.88	46.87	50.70
		(11.30)	(6.19)	(3.04)	(1.05)	(16.65)	(16.52)
	Low compr.	23.79	22.14	6.87	1.15	43.45	53.95
		(5.20)	(6.36)	(2.63)	(0.66)	(13.56)	(13.80)
	p diff.	.96	.75	.17	.56	.67	.69
String matching	High compr.	16.18	25.95	9.56	1.49	43.94	53.17
		(2.49)	(11.23)	(5.08)	(2.84)	(14.20)	(14.77)
	Low compr.	18.07	28.87	15.84	1.55	33.59	64.33
		(1.79)	(8.41)	(6.33)	(0.65)	(12.17)	(12.66)
	p diff.	*.11*	.58	**.04**	.96	*0.15*	*.13*

String matching program During comprehension of the string matching program, none of the high-comprehenders' fixation times on the areas of the visualization were significantly correlated with information types found in their summaries. High-comprehenders' fixation times on Code and Visualization areas were correlated neither with info-low nor with info-high types in any program. The correlation of low-comprehenders' time on Constants-area with info-high proportion approached p level of .1, (r(7)=-.64, *p=.120*). Information types contained in the summaries of low-comprehenders were not significantly correlated with time spent looking on the Code nor with the time looking at the Visualization. The distributions of the fixation times differed between the two performance groups in this case (table 3): low-comprehenders spent less time on Code, 21% more time on Visualization, and significantly more time on the Instances area.

5 Discussion

Visual strategies of programmers that use program visualization to aid comprehension depend on several factors. Two such factors, the target program and its actual visualization, seem to prevail over previous experience with programming [19]. Therefore, in the present study we began to investigate the differences in the visual behavior as related to the performance levels based on comprehension summaries. We focused on the interplays between the correctness of the comprehension summary, quality of the externalized mental models, successful comprehension strategies and related gaze patterns. Participants were post-hoc grouped according to whether their program comprehension summaries contained the information 1) about what the program does, 2) how the program solves the problem, and 3) about overall function and aim of the program.

Comprehension performance was measured as a correctness of description of the program function and procedure. Our results suggest that it seems not to be strongly related with the externalized information about mental models. In other words, presence of e.g. high-level information type in the comprehension summary does not indicate that the target program was correctly comprehended and vice-versa.

Analysis of externalized mental models indicates that high-comprehenders referred to higher-level information more, which is not a surprising finding. However, in the case of the string matching program, the less successful participants improved their performance in terms of higher-level understanding of the program so that there was not any significant difference in the proportions between the two groups.

Looking at the patterns of visual attention, our results offer one explanation: while high-comprehenders' visual strategies do not seem to be correlated with the level of abstraction found in their summaries, their performance is higher. In other words, it is hard to find common patterns in the visual strategies of high-comprehenders as their visual behavior seems to be rather individual. This conclusion seems to be in line with the findings of Crosby and Stelovsky [16].

During the comprehension of both programs, the attention of the high-comprehenders was more balanced between the different representations compared to the low-comprehenders. This seems to support the previous results about visual strategies during debugging [18]. It is surprising how the distribution of fixation times of low-comprehenders during binary-search resembles the overall distribution representing the high-comprehenders' behavior. While the performance differs, the times spent on different areas during comprehension of this program were similar. However, high-comprehenders, perhaps, had the abilities and knowledge to extract the information from different areas as necessary. Similarly distributed visual attention of the low-comprehenders then led to failed comprehension of the program. As seen from table 2, the more the low-comprehenders read the Code during animation of the binary search program, the more higher-level information was contained in their program summaries. At the same time, looking into the visualization decreased the proportion of the higher-level references in their mental models.

Let's contrast the previous with the behavior during the comprehension of the string-matching program. For many of the areas (Method, Instances, and also code), visual attention patterns of the two groups differed. For example, low-comprehenders paid more attention to the Visualization area, in particular, Instances were attended significantly more. We believe, that these strategies then led to the improvement in the proportion of higher-level information found in the low-comprehenders' summaries of string-matching program. To reliably explain this improvement and the difference, a further qualitative analysis, such as questioning of the participants, would be required.

To summarize the previous observations, we have found that during comprehension of one program the visual strategies of two performance groups were similar while their mental models differed. On the other hand, during comprehension of another program the visual strategies were different while the mental models were similar.

An alternative explanation for the finding could be that although the order of the programs was randomized in the original experiment, the post-hoc assignment into groups could result in an unbalanced order within the groups. However, this was not the case: the order of programs was balanced also for the post-hoc groups. It remains therefore most probable that the actual target programs and their respective visualizations played an important role in the differences found in comprehension and visual behavior.

While there has been found an evidence of the relation between the fixation patterns and cognitive processing, e.g. in reading [10], eye-movement data are generally difficult to interpret in the complex domains. This is partly due to the fact that in complex domains, such as programming, some information and processes are implicit and therefore are hard to represent explicitly. For example, during a constructor call the constructor of the super class is called even though this call would not be explicitly written into the program code. Programmers might understand the call even without visually attending the code of the constructor.

Could eye-tracking data help to explain the differences found in comprehension summaries? The presented findings suggest that gaze-patterns, at least partially, can predict information later found in mental models of programmers. For instance, the information shown in the Method and Instances areas expose higher-level information about program execution. Therefore, visually attending these areas shall have an effect on an increased awareness of the higher-level information related to execution. Similarly, attending Constants and Expression evaluation areas shall provide the programmer with low-level information. Our findings indicate, however, that looking at the Expression evaluation area might have decreased the proportion of high-level information of low-comprehenders. However, at the same time, the results also empirically manifest that just looking at visualization does not guarantee its correct understanding [24].

Although a promising technology, the gaze-related data, as used in this experiment and similar research, do not directly allow us to identify strategies that may lead to good comprehension. To overcome these problems, we are currently developing new measures and analysis techniques that would better reflect the cognitive processes involved in comprehension. For example, we employ the gaze to estimate the (gradual) *changes in importance* of the different representations to the programmer [25]. Thus, in future we can support the comprehenders efforts when the gaze is used in real-time.

There are, however, also other factors at play in determining good and poor comprehenders than just what structures of visualization they select to visually attend. For instance, according to related research, good comprehenders can combine their previous domain and programming knowledge with the information about the program in order to make sense of it [2]. It is quite possible that also in our study some of the more experienced participants could build good mental models even without paying attention to the visualization. In addition, it is also possible that more experienced participants failed to comprehend the programs and therefore they were classified into the low-comprehender group. In our future analysis we plan to take this possibility into account.

Furthermore, different learning styles can affect the comprehension process and certain types of learners might be more ready to comprehend programs with a help of an animation than others. However, gaze data may give indications of what kind of knowledge the comprehender tries to interpret.

5.1 Limitations and future work

The presented results have to be critically considered with a certain caution. The sample size was relatively small and the applied analysis requires a great number of correlations to be run. Therefore the generalizability of the conclusions could be limited. Further studies shall include a larger number of participants and validate or reject our findings.

Our results, however, provide an interesting starting point for further investigations. In our future work, we wish to more accurately classify the strategies leading to different mental representations of a program. If we can distinguish between good and poor comprehenders based on the gaze data or get indications about the abstraction level of their mental models, we might be able to support different comprehenders in their learning with an appropriately adapted support to foster comprehension.

In addition, we plan to improve the methodology introduced in this paper. First, the analysis of the comprehension summaries shall be conducted by several raters. In this way, we will be able to estimate an agreement between the raters and therefore add a validity measure for each of the correlations. The coding book provided in [22] and improved in [23] is comprehensive. The complete set of Good's information types will be analyzed, however, multivariate type of correlations or canonical correlations have to be used when the number of correlations grows. Finally, other gaze data, such as fixation duration and area switching, will be analyzed.

6 Conclusions

Eye-movement data indicate what structures of program visualization have been attended by programmers during comprehension. Program comprehension summaries contain information about mental models acquired during the comprehension. We have examined the relation between these two sources of data and we compared them with high- and low-comprehension outcomes.

We have not found any correspondence between program comprehension performance and structure of created mental models as indicated by information type analysis.

Although further studies are needed to achieve generalizability and higher validity, our data suggest important implications into the research of program comprehension. High-comprehenders divided their attention on the visualization and the code in a more balanced way. A few correlations between the eye-movement patterns and information types in the summaries of low-comprehenders have been found. When low-comprehenders' viewing times on different representations resembled those of high-comprehenders, they attempted to infer information on wrong abstraction levels from the visualization and the source code. This, in our opinion, negatively affected the low-comprehenders' results. When the fixation-time distributions and therefore the strategies of the two groups diverged and the low-comprehenders concentrated more on the visualization, the quality of their mental models improved. Yet, they failed to correctly comprehend the program. This implies that personal strategies might be more effective depending on the knowledge level of the comprehender.

Our long term aim is to develop systems that could assist the users in program comprehension and debugging by analyzing their gaze-data in real-time. This work takes the first steps into this direction.

References

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

2. Vans, A., von Mayrhauser, A., Somlo, G.: Program understanding behavior during corrective maintenance of large-scale software. Int. Journal of Human-Computer Studies **51** (1999) 31–70
3. Wiedenbeck, S., Fix, V., Scholtz, J.: Characteristics of the mental representations of novice and expert programmers: an empirical study. Int. Journal of Man-Machine Studies **39** (1993) 793–812
4. Ericsson, K.A., Simon, H.A.: Protocol analysis: Verbal reports as data. MIT Press, Cambridge, MA (1984)
5. Hundhausen, C.D., Douglas, S.A., Stasko, J.T.: A meta-study of algorithm visualization effectiveness. Journal of Visual Languages and Computing **13** (2002) 259–290
6. Ormerod, T.C.: Human cognition and programming. In Hoc, J.M., Green, T.R.G., Samurcay, R., Gilmore, D.J., eds.: The Psychology of Programming, Academic Press (1990) 63–82
7. Gugerty, L., Olson, G.M.: Comprehension differences in debugging by skilled and novice programmers. In: First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers. (1986) 13–27
8. Petre, M., Blackwell, A.: Mental imagery in program design and visual programming. International Journal of Human-Computer Studies **51** (1999) 7–30
9. Adelson, B., Soloway, E.: The role of domain experience in software design. IEEE Transactions on Software Engineering **11** (1985) 1351–1360
10. Just, M.A., Carpenter, P.A.: Eye fixations and cognitive processes. Cognitive Psychology **8** (1976) 441–480
11. Rayner, K.: Eye movements in reading and information processing: 20 years of research. Psychological Bulletin **124** (1998) 372–422
12. Goldberg, J., Kotval, X.P.: Computer interface evaluation using eye movements: Methods and constructs. International Journal of Industrial Ergonomics **24** (1999) 631–645
13. Jacob, R.J.K.: The use of eye movements in human-computer interaction techniques: what you look at is what you get. ACM Transactions of Information Systems **9** (1991) 152–169
14. Stein, R., Brennan, S.E.: Another person's eye gaze as a cue in solving programming problems. In: ICMI '04: Proceedings of the 6th Int. Conference on Multimodal Interfaces, New York, NY, USA, ACM Press (2004) 9–15
15. Torii, K., Matsumoto, K., Nakakoji, K., Takada, Y., Takada, S., Shima, K.: Ginger2: an environment for computer-aided empirical software engineering. IEEE Transactions on Software Engineering **25** (1999) 474–492
16. Crosby, M.E., Stelovsky, J.: How do we read algorithms? A case study. IEEE Computer **23** (1990) 24–35
17. Wiedenbeck, S.: Beacons in computer program comprehension. Int. Journal of Man-Machine Studies **25** (1986) 697–709
18. 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
19. 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
20. Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M.: Visualizing programs with Jeliot 3. In: Proceedings of the Working Conference on Advance Visual Interfaces (AVI 2004), ACM (2004) 373–376
21. 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

22. Good, J., Brna, P.: Program comprehension and authentic measurement: a scheme for analysing descriptions of programs. Int. Journal of Human Computer Studies **61** (2004) 169–185
23. Byckling, P., Kuittinen, M., Nevalainen, S., Sajaniemi, J.: 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). (2004) 170–184
24. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. CACM **38** (1995) 33–44
25. Bednarik, R., Tukianen, M.: An eye-tracking methodology for characterizing program comprehension. In: ETRA06: Eye Tracking Research and Applications, New York, NY, USA, ACM Press (2006) 125–132