In P. Romero, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 17 Pages 77 - 91

Short-Term Effects of Graphical versus Textual Visualisation of Variables on Program Perception

Seppo Nevalainen and Jorma Sajaniemi

University of Joensuu, Department of Computer Science,
P.O.Box 111, 80101 Joensuu, Finland,
seppo.nevalainen@cs.joensuu.fi,
WWW home page: http://www.cs.joensuu.fi/~snevalai/

Abstract. The empirical evaluation of program visualisation has been based mostly on observations of long-term effects of the program visualisation tools, while possible short-term effects of the visualisations and their relation to the long-term effects have been elided. In order to study short-term effects of visualisation of variables in a context where the long-term effects are already known, we conducted a controlled experiment, in which we investigated how a person targets her visual attention and what kind of a mental model she constructs, when variables are presented either textually or graphically. The results indicate clear differences in the targeting of visual attention between the visualisation tools: With the graphical tool, the participants targeted their visual attention to variables much more than with the textual tool. With the graphical tool, the increase of visual attention to variables increased the proportion of high-level information in program summaries and decreased the proportion of low-level code-related information.

1 Introduction

Learning to program is a difficult task for many students. One reason for this is that programs deal with abstract entities—formal looping constructs, pointers going through arrays etc.—that have little to do with everyday issues. Methods and techniques that help students to better understand and conceptualise these abstract entities and their behavior can be used to enhance learning elementary programming. Visualisations can be used for this purpose, e.g., to illustrate expert programmers' reasoning processes to the novice [1] or to make programming language constructs and program constructs more comprehensible [2, 3].

What, then, should be chosen as the focus of a visualisation, and how it should be presented to the viewer? Variables are central to the comprehension of computer programs. Programs consist of variables, operations on variables, and larger program constructs, such as functions, classes, and modules. In one study [4], information about variables was the most frequent information need type among professional maintenance programmers. Several taxonomies and frameworks [5–9] have been presented to aid designers and evaluators of software visualisation tools and visual programming environments in identifying the essential aspects of visualisations. This information can be utilised in searching answers for the question how information should be presented in

order for a visualisation to be effective. Our emphasis will be on issues concerning the visualisation of program variables.

To verify the effectiveness of a visualisation tool, it needs to be properly evaluated. Empirical evaluation of program visualisations has been based mostly on post-tests or pre- versus post-tests of participants' performance. These evaluations have resulted in a body of evidence suggesting that visualisations can have beneficial long-term effects on learning, when designed and used properly [2, 10–12]. Evaluation of post-test or pre- versus post-test performance of participants does not, however, provide clear insight into the possible short-term effects of visualisations and their relation to the long-term effects.

In order to study these issues more rigorously, we conducted an experiment, in which we studied two different visualisation tools for presenting program variables and their execution-time behavior. The first tool, PlanAni program animator[13], presents pictorial metaphors for variables, operations on variables are animated, and information concerning the roles of the variables [14] is incorporated into the visualisation. The second tool, Turbo Pascal programming environment, provides textual representations of variables, operations on variables simply replace the value of the variable in the representation, and no role information is present. Previous studies [13, 15] have reported the differences in long-term effects of the use of these two tools.

Visualisation can be effective only if it gets viewers' visual attention. In order to determine the level of viewers' visual attention on program code and on visualisation of variables with the two visualisation tools, the locations of the participants' gaze on the screen were measured. Possible differences in the participants' mental models of the studied programs between the two visualisation tools were investigated by analysing participants' program summaries. In order to control possible individual differences between the participants, field-indepence of each participant was also measured.

This paper is organised as follows. First, we provide some background by reviewing the literature on visualisation of variables in existing program visualisation tools in Section 2. Section 2 also includes an introduction to the roles of variables and their visualisation. In Section 3, the experiment is described, and its results are presented and discussed. Section 4 contains the conclusions.

2 Background

In this Section, we will discuss different manners of representation of variables in program visualisation. First, we will take a look at the visualisation of variables in general and then we will focus on the representation of variables in the graphical environment of PlanAni, which is paired with the textual environment of Turbo Pascal in our experiment.

2.1 Visualisation of Program Variables

Visualisation can be characterised as a formation of a mental image on the basis of some sensory input [16]. In program visualisation tools, this sensory input often means pictorial metaphors, i.e., static or animated images. Pictorial metaphors produce mental

images that can be helpful, irrelevant, or even harmful in accomplishing the task(s) being carried out.

What sort of visualisation is suitable for which information and which uses by which users? This multifaceted question is discussed by Petre et al. [1] who give some idea on the complexity that the design of a program visualisation tool involves. General frameworks and taxonomies [5–9] have been presented in order to help designers' alleviate this complexity, but they do not provide definite answers for feature selection.

In this paper, the design of visualisation of variables is considered on the following axes:

- Predefined visualisations versus user-defined visualisations.
- Programming language level visualisations versus visualisations of higher level programming constructs.
- Targeting of animation on a variable versus on the visualisation of the variable versus on both.

Program visualisation tools can be divided into two categories: semi-automatic tools and hand-crafted tools [13]. This division is based on how much a program visualisation tool allows the user to influence the visual appearance of variables. In semi-automatic tools, users select visualisations for variables from a set of ready-made visualisations. In hand-crafted tools, users make choices to reflect the value of a variable by choosing for example the appropriate size, color, and orientation for the visualisation. Program visualisation tools can use also predefined visualisations selected by the designer of the visualisation. The main effect of allowing users to participate in deciding the visual appearance of the visualisations is the increased interaction between the visualisation tool and the user; there is no guarantee of the appropriateness of the visualisation.

Many of the current visualisation tools represent variables and operations on variables in the program or programming language level. That is, they show what variables a program includes and the change of the values of these variables during program execution, treating each variable as an individual having at most programming language level abstractions (such as the type of the variable). According to Petre and Blackwell [17], visualisations should not work in the programming language level because within-paradigm visualisations, i.e., those dealing with programming language constructs, are uninformative. Instead of treating each variable as a separate entity, expert programmers place variables into different classes according to their stereotypic behavior and use [18]. These stereotypic features of variables are an example of higher level program constructs that can be utilised in visualisations.

When animation is used in a visualisation, it can be focused on different aspects of the visualisation. Sajaniemi and Stützle [19] introduce three possible foci: the variable, the metaphor used as the visualisation of the variable, or combination of the previous two. The effects of different targeting of animation have not been verified, but possible effects have been suggested. If the focus is on the metaphor, the animation probably helps to recognise the metaphor or gives more enjoyment to the viewer. If the focus is on the properties of the variable, this presumably helps to deepen the understanding of that variable. If the emphasis is on both the variable and the metaphor, the connection between the two is emphasised, and the analogy between them may become more evident.

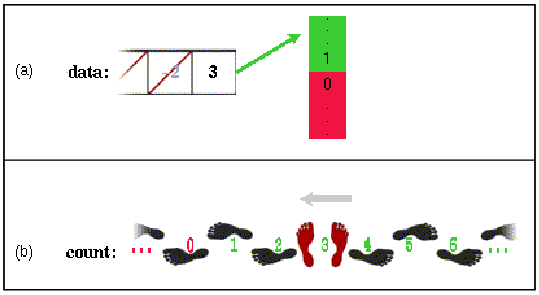

Fig. 1. Two variable roles, most-recent holder (data) and stepper (count), and animations representing the comparison operation "$some\_variable > 0$".

2.2 Visualising Roles of Variables

To facilitate teaching introductory programming, Sajaniemi has developed a theory of the roles of variables [14]. This theory is based on the notion of variable plans, which represent stereotypic uses of variables [20, 21]. Ten roles are sufficient to cover practically all variables in novice-level procedural programs. These roles have also been found to belong to professional programmers' tacit knowledge [22]. On the basis of this theory, Sajaniemi and Kuittinen [13] have introduced a program animator, PlanAni.

In Planani, visualisations of variables are predefined: Each variable has a role image which is used also for the animation of operations on the variable. For example, the role images of two roles, most-recent holder and stepper, and animations representing the comparison operation "$some\_variable > 0$", are shown in Figure 1. The role images represent the salient, stereotypical features of variables' behavior. Animations in PlanAni are focused on the target, i.e., role-like behavior of variables. Stützle and Sajaniemi [23] have evaluated role images of PlanAni empirically and found that the role images enhanced learning of the roles when compared with neutral control images.

Long-term effects of the visualisation of variable roles have been analysed in a classroom experiment [24]. Both Sajaniemi and Kuittinen [24], and Byckling and Sajaniemi [15], have found results suggesting that visualisation of variable roles with PlanAni has positive long-term effects on learning when compared with textual Turbo Pascal environment. Short-term effects and their relation to long-term effects are the focus of this and our further experiments.

3 Experiment

In order to study short-term effects of variable visualisations, we conducted an experiment where two different visualisation tools for presenting information about variables and their values during program execution were used. The tools were used in the classroom experiment [24], in which the long-term effects of the visualisations were investigated. The current study concentrates on possible differences in the locations of participants' visual attention and in participants' mental models of the studied programs between the two visualisation tools.

For control reasons, the level of field-independence of each participant was measured. Witkin et al. [25] define field-dependence and field-independence as follows: ”in a field-dependent mode of perceiving, perception is strongly dominated by the overall organization of the surrounding field, and parts of the field are experienced as ”fused”. In a field-independent mode of perceiving, parts of the field are experienced as discrete from organized ground.” Field-independence has been found to correlate positively with learning to program [26], especially in computerised text-based and web-based environments [27]. Parkinson et al. [28] have shown that the difference in performance between field-dependent and field-independent learners in computerised text-based and web-based environments can be diminished by accommodating field-dependence in the design of the environments.

The experiment consisted of 4 phases. In the first phase, the participants were asked to perform a test measuring participants' level of field-independence. In the second phase, the participants studied a recap material on roles of variables. The third phase consisted of viewing Pascal programs with the visualisation tools and of writing down program summaries. In the fourth phase, the participants filled a questionnaire about the visualisation tools.

3.1 Method

The experiment was a within-subject design with one independent variable (the visualisation tool) and two dependent variables (locations of the participant's gaze and the program summary provided by the participant). Locations of gaze were recorded using an eye-tracking camera [29], and program summaries were analysed using Good's program summary analysis scheme [30]. The level of field-independence of the participants was measured using Group Embedded Figures Test (GEFT) [25]. The order of the visualisation tools and the order of the studied programs were counterbalanced.

Participants Twelve participants, 7 male and 5 female, took part in the experiment. The participants were students who had taken an introductory programming course facilitating the roles of variables and continued their studies 1-2 years thereafter.

Materials In the first phase, participants' level of field-independence was measured using GEFT test set [25].

In the second phase, written material from an earlier experiment [19] was used. The material consisted of descriptions of all roles and examples of their use. It included also a practice material consisting of three small Pascal programs with 14 variables, whose roles participants were asked to determine.

In the third phase, participants studied four simple Pascal programs. The programs were short (11-29 lines, empty lines omitted) and similar to the more difficult programs used in the introductory programming course of the earlier classroom experiment [24]. Participants entered predefined inputs for the programs. The use of fixed inputs enabled a participant to focus her attention to understanding the program, instead of wondering what inputs would be proper to the programs.

The visualisation tools used were the PlanAni programming animator (version 0.55) and the Turbo Pascal programming environment (version 5.5). In PlanAni (Figure 2), visualisations are graphical, operations on variables are animated, and information concerning the roles of variables is incorporated into the visualisations. Variable visualisations are located on the right side of the program code. PlanAni displays also notifications of each program action and has a separate area for input and output. In Turbo Pascal (Figure 3), visualisations are textual, operations on variables simply replace the old value of the variable with the new value, and no role information is presented. Variable visualisations are located below the program code. Turbo Pascal displays no notifications, and input and output are handled through command prompt. In the Turbo Pascal environment, watches displaying each variable and its values during execution were initialised in advance, and they served as textual visualisations of variables. Both visualisation tools display code and variables and were prepared so that participants were able to execute each program once, step by step. This limitation was used because the tools differed in many other aspects and we wanted to minimise differences having an influence on the participants.

In the fourth phase, participants were asked to evaluate the visualisation tools with an evaluation form including Likert scale questions and open questions about the tools and their use. In the Likert scale questions, participants were asked to use a scale of 1-5 (1 = totally disagree, 5 = totally agree) to statements concerning five characteristics of the visualisations: originality, pleasure, salience, understandability, and usefulness. For example, the understandability of the visualisations was evaluated by proposition ”I found this representation easy to understand”. These characteristics were derived from experiments carried out by Hübscher-Younger and Narayanan [31] who used them to characterise student visualisations of algorithms. In the open questions, participants were asked to report what issues the two visualisations did and did not highlight. The evaluation form included also a possibility for free commentary.

Procedure Participants were run individually. Each participant’s level of field-independence was measured with the GEFT test consisting of three phases that lasted 2 minutes, 5 minutes, and 5 minutes. After this, the participant was given 15 minutes to study the roles of variables recap material and perform the practice task. Then, after a short break, the participant was seated in front of a computer monitor that has an eye-tracking camera embedded in the panels. The procedure of measuring the movement of her eyes was explained to the participant, and she was advised about the locations of all available information on the screen for both visualisation tools. The participant had then an unlimited time to study each program. After the participant had finished studying a program, the program was dismissed from the screen, and she was instructed to give a written description of the program. Again, the time to do this was not limited, and the participant was not instructed in any way on what the program description should comprise of. The first two programs were shown with one visualisation tool and the next two with the other tool. The first program with each visualisation tool was used to familiarise the participant with the tool, and data from the second program only was analysed. When all four programs had been studied, the participant was asked to evaluate the visualisation tools.

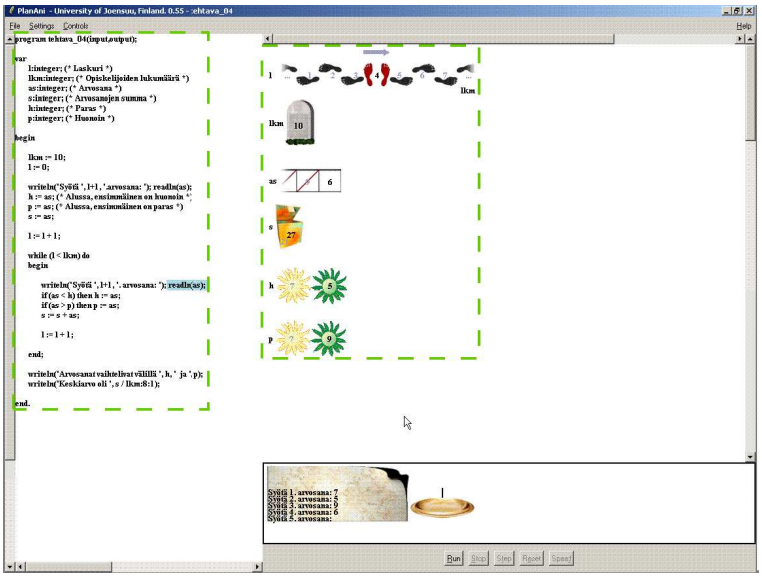

Fig. 2. User interface of the PlanAni program animator (Dashed rectangles represent the code area and the variable visualisation area used in the analysis of gaze locations).

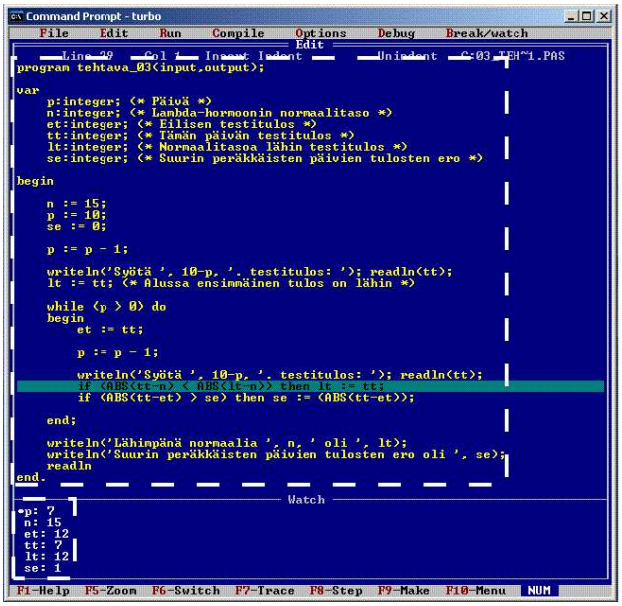

Fig. 3. User interface of the Turbo Pascal programming environment (Dashed rectangles represent the code area and the variable visualisation area used in the analysis of gaze locations).

3.2 Results

GEFT test results of participants' levels of field-independence are shown in Table 1. In the GEFT test, higher score means higher level of field-independence, and the theoretical maximum is 18.

Table 1. Results of GEFT test measuring participants' levels of field-independence.

	n	Min	Max	Mean	SD
GEFT score	12	7	18	14.75	3.22

The participants used on an average 26 minutes and 18 seconds to study a program with PlanAni. Standard deviation was 5 minutes and 58 seconds. With Turbo Pascal, mean time to study a program was 6.08 (SD 1.57). Due to the difference in the speed of the animation between the two tools, the minimum time it takes to view the shorter of the two analysed programs with the tools is 11.30 for PlanAni and 1.00 for Turbo Pascal.

Table 2. Mean proportions of viewing times on the three areas of the screen.

Code	Screen Area	Condition			
		PlanAni		Turbo Pascal	
		Mean	SD	Mean	SD
COD	Code ***	20.36	3.27	38.75	2.13
VAR	Variables ***	15.43	3.59	3.11	2.38
OTH	Other ***	64.21	2.89	58.14	2.52

For the purpose of the analysis, the screen was divided into three areas. The code area and the variable area were formed by taking the smallest bounding box that includes the symbols used in the code or the variable visualisations. These areas are illustrated by the dashed rectangles in Figures 2 and 3. Other parts of the screen formed the third area. A two-way within-subject Analysis of Variance was carried out. The ANOVA on absolute viewing times showed that there was a significant main effect of visualisation tool ($F(1,9) = 156.956$, $p < 0.001$), and of screen area ($F(2,9) = 78.125$, $p < 0.001$), and also a significant two-way interaction of visualisation tool and screen area ($F(2,9) = 55.984$, $p < 0.001$). The mean proportions of viewing times on these three areas are presented in Table 2. Paired t-test with Bonferroni correction was used for follow-up testing. The difference between PlanAni and Turbo Pascal is significant in the proportional viewing time on code ($t = -17.036$, $df = 11$, $p < 0.001$), variables ($t = 8.721$, $df = 11$, $p < 0.001$), and other parts of the screen ($t = 5.708$, $df = 11$, $p < 0.001$).

In order to study participants' mental models of the studied programs, we used Good's program summary analysis scheme [30] that consists of two classifications: one

based on information types (IT) and the other based on object descriptions (ODC). The information types classification is used to code summary statements on the basis of the information types they contain. Table 3 contains the distribution of information type statements in each condition. The object descriptions classification looks at the way in which objects are described. Table 4 contains the distribution of object description statements in each condition. No statistically significant differences between the conditions were found in information types or object description classifications.

We analysed the distribution of domain versus program information in participants' program summaries further by using a similar strategy as Sajaniemi and Kuittinen [24]. We sorted program summaries into three types depending on the amount of domain versus program statements in object descriptions. Summaries with at least 67% domain statements (indirect and unclear statements excluded) were called *domain-level summaries*, summaries with at least 67% program and program only statements were classified as *program-level summaries*, and all others were called *cross-referenced summaries* because they had a more even distribution of domain and program information. The number of cross-referenced summaries was two in PlanAni condition and four in Turbo Pascal condition. This difference is not statistically significant (Fisher's exact test).

Table 3. Mean proportions of IT categories used in program summaries.

Code	Information Type	Condition			
		PlanAni		Turbo Pascal	
		Mean	SD	Mean	SD
FUN	Function	14.73	32.96	8.33	21.62
ACT	Actions	17.98	18.12	16.08	16.98
OPE	Operations	13.16	15.42	10.64	12.83
SHI	State-high	4.42	5.15	2.38	5.79
SLO	State-low	3.84	5.92	3.68	6.18
DAT	Data	36.69	25.56	41.40	25.41
CON	Control	3.70	6.35	6.86	10.88
ELA	Elaborate	3.45	5.69	8.15	17.34
MET	Meta	0.52	1.79	0.38	1.30
IRR	Irrelevant	1.13	3.93	1.07	2.62
UNC	Unclear	0.00	0.00	0.00	0.00
INC	Incomplete	0.00	0.00	0.00	0.00
CUT	Continuation	0.00	0.00	0.38	1.30
HIG	FUN+ACT+SHI+DAT	73.80	25.54	68.18	24.55
LOW	OPE+SLO+CON	21.08	22.97	21.86	19.93
OTH	100-HIG-LOW	5.12	8.86	9.96	17.61
HIP	HIG / (HIG+LOW) * 100	77.35	23.78	76.08	22.11

Participants' evaluation of the visualisation tools is presented in Table 5. The difference between PlanAni and Turbo Pascal is significant in both originality ($t = 7.374$, $df = 11$, $p < 0.001$) and salience ($t = 4.690$, $df = 11$, $p = 0.001$).

Table 4. Mean proportions of ODC categories used in program summaries.

Code	Object Description Category	Condition			
		PlanAni		Turbo Pascal	
		Mean	SD	Mean	SD
PON	Program only	0.98	3.41	1.89	6.55
PRO	Program	1.96	6.78	2.25	6.56
PRR	Program—real-world	18.19	26.10	16.22	18.74
PRD	Program—domain	2.19	4.23	1.81	6.26
DOM	Domain	74.92	29.41	77.13	22.23
IND	Indirect reference	1.78	2.64	0.69	2.40
UNO	Unclear	0.00	0.00	0.00	0.00

Table 5. Participants' evaluation of different characteristics of the two visualisation tools (scale 1-5); the best is 5.

Characteristic	Condition			
	PlanAni		Turbo Pascal	
	Mean	SD	Mean	SD
Originality ***	3.92	0.79	2.00	0.60
Pleasure	2.58	1.00	3.42	1.00
Salience **	4.00	0.43	3.00	0.74
Understandability	4.25	0.75	3.50	0.91
Usefulness	3.00	1.04	2.50	0.80

The correlations between participants' levels of field-independence and the dependent variables—the proportions of time used for viewing the program variables, and the proportions of different information types and object description categories in participants' program descriptions—were analysed using the Pearson correlation coefficient. Variables having statistically significant correlation with proportion of time used for viewing the visualisations of program variables (VAR) are shown in Table 6. There were no statistically significant correlations between proportion of time used for viewing the code and any of the variables.

Table 6. Variables having statistically significant correlation with time used for viewing the visualisations of program variables.

Correlation	Condition	
	PlanAni	Turbo Pascal
VAR versus GEFT score	0.688 *	-0.071
VAR versus HIP	0.601 *	-0.084
VAR versus OPE	-0.655 *	0.042
VAR versus SLO	-0.725 **	0.025
VAR versus PRR	-0.445	-0.747 **

In PlanAni condition, the Pearson correlation coefficient between proportion of variable viewing and the GEFT-score is $r = 0.688$, the two-tailed probability for a correlation of such magnitude to occur by chance being statistically significant ($t = 3.001$, $df = 10$, $p = 0.0133$). In PlanAni, correlation is statistically significant also between proportion of variable viewing and high-level IT-descriptions (HIP) ($r = 0.601$, $t = 2.377$, $df = 10$, $p = 0.0388$), proportion of variable viewing and operation level IT-descriptions (OPE) ($r = -0.655$, $t = -2.741$, $df = 10$, $p = 0.0208$), and proportion of variable viewing and state-low level IT-descriptions (SLO) ($r = -0.725$, $t = -3.331$, $df = 10$, $p = 0.0076$). In Turbo Pascal, statistically significant correlation occurs between proportion of variable viewing and program—real-world object descriptions (PRR) ($r = -0.747$, $t = -3.556$, $df = 10$, $p = 0.0052$).

3.3 Discussion

The purpose of this experiment was to investigate how a person targets her visual attention, and what kind of a mental model she constructs of a computer program, when the program is presented using a program visualisation tool. The experiment is first in a series of experiments that will study in detail the effects of the visualisation of roles of variables in PlanAni. Two completely different visualisation tools were selected for the current experiment in order to bring forth clearly different effects that different visualisation tools might produce and to provide this way a starting point for a more detailed investigation in future. Furthermore, the long-term effects of these two tools have been studied earlier [13, 15].

The results indicate that participants spent more time viewing both the code and the variables with PlanAni than they did with Turbo Pascal. Part of this can be explained

by the difference in the speed of the animation. Another explaining factor may be the difference in the graphical richness and amount of details between the two tools.

The variable visualisations were viewed proportionately more with PlanAni than with Turbo Pascal ($p < 0.001$) which means that the animation tool has an effect on visual attention. One explaining factor is the location of animation: in PlanAni, most animations appear within the variable visualisations, whereas in Turbo Pascal they appear in the code area. The effect of other factors, e.g., the pleasantness of the images must be studied separately. The other area of the screen was viewed proportionately more with PlanAni than with Turbo Pascal ($p < 0.001$). This was probably because the area was substantially larger in PlanAni, and because it displayed input and output of the program to the viewer constantly, instead of displaying them only in command prompt.

Program summaries were used to study the mental models of the participants. No statistically significant differences were found between the two tools. However, in PlanAni the proportion of variable viewing correlated positively with high-level information ($r = 0.601$) and negatively with operations ($r = -0.655$) and state-low ($r = -0.725$) in program summaries. Thus the increase of visual attention in the variable visualisation area increased high-level data-related information; and the increase of visual attention to the code area increased low-level code-related information. In Turbo Pascal these effects could not be found. Thus either the smaller absolute time increase was not sufficient to cause changes in mental model or the Turbo Pascal interface did not provide the information required for the high-level mental model because it lacks role information.

In Turbo Pascal, proportion of variable viewing correlated negatively with program-real world object descriptions ($r = -0.747$; PlanAni $r = -0.445$). The program-real world object descriptions typically contained expressions such as “value” and “number”, which were used in a similar way as program object descriptions in describing the low-level operations of the programs. With both tools the increase of visual attention in variable visualisations decreased the participants use of low-level descriptions of the programs.

PlanAni has earlier been found to have positive long-term effects on programming skills and content of mental models [13, 15], but in this experiment such an overall effect could not be found. In addition to the location of visual attention, a person’s mental model is influenced by other factors also. Hübscher-Younger and Narayanan [31] have used six characteristics of visualisation tools and studied their effect on learning. They found pleasure and salience to be the two most important characteristics influencing learning. We asked our participants to evaluate both visualisation tools with five of these characteristics: originality, pleasure, salience, understandability and usefulness. The evaluation form included also open questions and a possibility for free commentary. In our experiment, participants judged PlanAni to be more salient ($p = 0.001$) than Turbo Pascal, but also more unpleasant ($p = 0.096$). One possible reason for the small effect on mental models in this experiment is that even though PlanAni was judged more salient, it was also found unpleasant to use.

On the basis of the open questions, it is obvious that the unpleasantness of PlanAni was mostly due to the slowness of the animation. Eight participants commented negatively about the slowness of PlanAni, and four commented positively about the fast

use of Turbo Pascal. The salience of PlanAni was contributed mostly to the illustration of variables' roles and tasks in the program. These were commented positively by the participants five times (roles of the variables) and four times (tasks of the variables). In some of the free commentary by the participants, PlanAni was deemed to be appropriate for teaching elementary programming, not for intermediates. Because we had 2nd and 3rd year students in the experiment, the programs were easy for the participants and therefore properties highlighted by animation may have not shown up in the program summaries. This can also partly explain the unpleasantness the participants felt in using PlanAni, a tool designed for true novices.

The proportion of variable viewing correlated with the GEFT score in PlanAni ($r = 0.688$), but not in Turbo Pascal ($r = -0.071$). This can be explained by the difference in graphical richness between the two visualisation tools. Following the textual visualisation of Turbo Pascal does not require the viewer to be able to separate items from organised perceptual field in the same way as with PlanAni, which uses colorful graphical images and animations. This is consistent with previous experiments [26, 27] that have studied the relationship between field-independence and learning. Thus, the level of field-independence influenced the targeting of visual attention, which influenced the mental model being constructed. Therefore, the level of field-independence has direct implications on the usefulness of visualisations.

4 Conclusions

We have studied how a person targets her visual attention, and what kind of a mental model she constructs concerning a computer program, when the program and especially its variables are presented using either a textual or a graphical program visualisation tool. PlanAni program animator uses role images and animations on these images to highlight program variables, while Turbo Pascal displays variables and their values textually and without role information.

The results indicate that visual attention of the participants was targeted on the variable visualisations clearly more with PlanAni than with Turbo Pascal. In PlanAni, the increase of visual attention to variables increased the proportion of high-level data-related information in program summaries and decreased low-level code-related information, thus effecting the mental models of the participants. In Turbo Pascal, these effects could not be found. Moreover, with PlanAni the proportion of variable viewing correlated positively with the level of field-independence. Thus field-independent students gain more by the graphically rich PlanAni program animator than field-dependent students.

There exists classifications that could be applied to the detailed investigation of different graphical factors' contribution to the overall attractiveness of visualisations and their effects on visual attention and on mental models. For example, Bertin [32] has introduced eight visual variables that can be identified in an image; Brown [33] has introduced an algorithm animation taxonomy; and Sajaniemi and Stützle [19] have introduced three different possibilities for targeting animation in visualisations. This experiment established that there exists differences in short-term effects of different vi-

sualisations and works as a starting point for further studies, in which we will investigate the short-term effects of visualisation in more detail.

Acknowledgments

This work was supported by the Academy of Finland under grant number 206574.

References

1. Petre, M., Blackwell, A., Green, T.R.G.: Coqnitive questions in software visualisation. In Stasko, J.T., Domingue, J., Brown, M.H., Price, B.A., eds.: Software Visualization – Programming as a Multimedia Experience, The MIT Press (1998) 453–480
2. 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
3. Mulholland, P.: A principled approach to the evaluation of SV: A case study in Prolog. In Stasko, J.T., Domingue, J., Brown, M.H., Price, B.A., eds.: Software Visualization – Programming as a Multimedia Experience, The MIT Press (1998) 439–451
4. von Mayrhauser, A., Vans, A.M.: Industrial experience with an integrated code comprehension model. Software Engineering Journal **10** (1995) 171–182
5. Myers, B.: Taxonomies of visual programming and program visualisation. Journal of Visual Languages and Computing **1** (1990) 97–123
6. Stasko, J., Patterson, C.: Understanding and characterizing software visualization systems. In: Proceedings of the 1992 IEEE Workshop on Visual Languages, IEEE Computer Society Press (1992) 3–10
7. Price, B., Baecker, R., Small, I.: A principled taxonomy of software visualisation. Journal of Visual Languages and Computing **4** (1993) 211–266
8. Green, T.R.G., Petre, M.: Usability analysis of visual programming environments: A 'cognitive dimensions' framework. Journal of Visual Languages and Computing **7** (1996) 131–174
9. Ainsworth, S.E., Labeke, N.V.: Using a multi-representational design framework to develop and evaluate a dynamic simulation environment, Dynamic Information and Visualisation Workshop (2002)
10. Byrne, M.D., Catrambone, R., Stasko, J.T.: Evaluating animations as student aids in learning computer algorithms. Computers & Education **33** (1999) 253–278
11. Hansen, S.R., Narayanan, N.H., Schrimpsher, D.: Helping learners visualize and comprehend algorithms. Interactive Multimedia Electronic Journal of Computer-Enhanced Learning **1** (2000)
12. Kann, C., Lindeman, R.W., Heller, R.: Integrating algorithm animation into a learning environment. Computers & Education **28** (1997) 223–228
13. Sajaniemi, J., Kuittinen, M.: Visualizing roles of variables in program animation. Information Visualization **3** (2004) 137–153
14. Sajaniemi, J.: An empirical analysis of roles of variables in novice-level procedural programs. In: Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02), IEEE Computer Society (2002) 37–39
15. Byckling, P., Sajaniemi, J.: Using roles of variables in teaching: Effects on program construction, Accepted to the the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2005) (2005)
16. Price, B., Baecker, R., Small, I.: An introduction to software visualization. In Stasko, J.T., Domingue, J., Brown, M.H., Price, B.A., eds.: Software Visualization – Programming as a Multimedia Experience, The MIT Press (1998) 3–27

17. Petre, M., Blackwell, A.F.: Mental imagery in program design and visual programming. International Journal of Human-Computer Studies **51** (1999) 7–30
18. Sajaniemi, J., Navarro Prieto, R.: An investigation into professional programmers' mental representation of variables. In: 13th IEEE International Workshop on Program Comprehension (IWPC 2005). (2005)
19. Sajaniemi, J., Stützle, T.: Evaluation techniques for animated software visualization metaphors (Submitted)
20. Ehrlich, K., Soloway, E.: An empirical investigation of the tacit plan knowledge in programming. In Thomas, J.C., Schneider, M.L., eds.: Human Factors in Computer Systems. Norwood, NJ: Ablex Publishing Company (1984) 113–133
21. Rist, R.S.: Knowledge creation and retrieval in program design: A comparison of novice and intermediate student programmers. Human-Computer Interaction **6** (1991) 1–46
22. Sajaniemi, J., Navarro Prieto, R.: Roles of variables in experts' programming knowledge, Accepted to the the 17th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2005) (2005)
23. Stützle, T., Sajaniemi, J.: An empirical evaluation of visual metaphors in the animation of roles of variables. Informing Science Journal **8** (2005) 87–100
24. Sajaniemi, J., Kuittinen, M.: An experiment on using roles of variables in teaching introductory programming. Computer Science Education **15** (2005) 59–82
25. Witkin, M.A.: A Manual for the Embedded Figures Test. Consulting Psychologists Press (1971)
26. Mancy, R., Reid, N.: Aspects of cognitive style and programming. In Dunican, E., Green, T., eds.: Proceedings of the Sixteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004). (2004) 1–9
27. Parkinson, A., Redmond, J.A.: Do cognitive styles affect learning performance in different computer media? In: The 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002), Association for Computing Machinery (2002) 39–43
28. Parkinson, A., Redmond, J.A., Walsh, C.: Accommodating field-dependence: A cross-over study. In: The 9th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2004), Association for Computing Machinery (2004) 72–76
29. Tobii: User Manual - Tobii Eye Tracker, Clearview Analysis Software. Tobii Technology AB (2004)
30. Good, J.: Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension. PhD thesis, University of Edinburgh (1999)
31. Hübscher-Younger, T., Narayanan, N.H.: Dancing hamsters and marble statues: Characterizing student visualizations of algorithms. In: ACM 2003 Symposium on Software Visualization (SoftVis 2003), Association for Computing Machinery (2003) 95–104
32. Bertin, J.: Semiology of Graphics. University of Wisconsin Press (1983)
33. Brown, M.: A taxonomy of algorithm animation displays. In Stasko, J.T., Domingue, J., Brown, M.H., Price, B.A., eds.: Software Visualization – Programming as a Multimedia Experience, The MIT Press (1998) 35–42