

Effectiveness of Program Visualization: A Case Study with the ViLLE Tool

*Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila,
and Tapio Salakoski*
University of Turku, Turku, Finland

temira@utu.fi; milaak@utu.fi; ertaka@utu.fi; sala@utu.fi

Executive Summary

Program visualization is one of the various methods developed over the years to aid novices with their difficulties in learning to program. It consists of different graphical – often animated – and textual objects, visualizing the execution of programs. The aim of program visualization is to enhance students' understanding of different areas of program execution. Typical program visualization techniques include code highlighting, visualization of the call stack, and presenting information on variables. Despite the large number of studies performed on program visualization, little is known about the effects of such systems on learning.

We have developed a program visualization tool called ViLLE, with the main objective of offering an environment for students to study the execution of example programs – whether written by students themselves or prepared by the teacher – and explore the changes in the program state data structures. A key feature of ViLLE is language independency, including parallel execution of a program in two different languages and the ability to define new languages. ViLLE also provides role information of program variables and supports the design and use of interactive pop-up questions.

In this paper, we report and discuss the results of a study on the effectiveness of ViLLE. The research was conducted on university students in their first programming course. Students participated in a two hour session in a computer class, where they were randomly divided into two groups. The control group used only traditional textual material during the session, whereas for the treatment group, the same material was extended with interactive examples using ViLLE. With this research setting, we tried to answer two research questions: “Does ViLLE help students in learning to program?”, and “Is there any difference in learning when previous programming experience is taken into account?” We found some support for a positive answer to the first question, although we couldn't fully reject the null hypothesis. For the second question, we obtained solid evidence that ViLLE enhances the learning of students with no prior programming experience substantially, so that the statistical differences between the novices and the more experienced learners disappeared as a result of

a single training session. This indicates that program visualization indeed improves novice students' learning.

Keywords: program visualization, novice programmers, effectiveness of visualization, programming, programming learning, programming teaching.

Material published as part of this publication, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact Publisher@InformingScience.org to request redistribution permission.

Introduction

Learning to program is not an easy task. According to multi-national studies published in recent years, students have problems in writing program code (McCracken et al., 2001), in reading and tracing skills (Lister et al., 2004), and in designing software (Tenenbergs et al., 2005). Since constructing and even understanding computer programs have proven to be a highly non-trivial task for most learners, various techniques and means have been suggested to aid the learning process of beginner programmers. Visualization – generally defined as presenting the execution of program or algorithm with graphical components – is one of these. According to Ben-Ari (2001) visualization includes everything even remotely graphical, from complex animations to indentation of program blocks, and for the effective use of visualizations, the textual and graphical descriptions have to be synchronized. Hyrskykari (1993) states that visualizations can be useful in providing learning models that can be used in linking new information with old knowledge.

Program visualization is a research area that studies ways of visually assisting learners in understanding behaviour of programs. The visualization of programs can be either dynamic or static. Dynamic program visualization tools visualize execution of programs. They usually show how the execution of programs progresses by highlighting parts of the code under execution and by visualizing changes in variable states. An example of a dynamic program visualization tool is Jeliot3 (Moreno, Myller, Sutinen, & Ben-Ari, 2004). Static visualization tools visualize program structures and relations between program objects. An example of a popular static program visualization tool is BlueJ (Kölling, Quig, Patterson, & Rosenberg, 2003).

We have recently developed a dynamic program visualization tool called ViLLE (Rajala, Laakso, Kaila, & Salakoski, 2007). ViLLE is a language-independent visualization tool aimed at providing a more abstract view of programming. The tool can be utilized both in lectures and for independent learning. It has a built-in syntax editor, with which the user can add new languages to the tool or modify the syntax of the built-in languages, currently including e.g. Java, C++, and a pseudo language. The visualizations can be viewed in any of the (user or pre-) defined languages. To emphasize the language independency, ViLLE has a parallel view in which the execution of a program and the program code itself can be viewed simultaneously in two languages. While the execution progresses, the user can observe program outputs and changes in variable values. In addition, to enhance the effectiveness and clarity of the visualization, there is an automatically generated textual description of each code line. The description also includes information about the roles of variables (Sajaniemi, 2002). However, according to Nikula, Sajaniemi, Tedre, and Wray (2007), to get the most benefit from the roles of variables, they should be employed in all aspects of teaching.

The goal of this paper is to find out what kind of effects ViLLE has on programming learning with following research questions: “Does ViLLE help students in learning to program?” and “Is there any difference in learning when previous programming experience is taken into account?”. To study these questions, we conducted a study in the first programming course at the University of Turku, Finland, in fall 2007.

This paper has the following structure. In the next section we consider previous work on program and algorithm visualization. In the third section, ViLLE and its key features are described. The research design and results are presented in the fourth and fifth sections, respectively. These are followed by a section in which the results are discussed and, finally, conclusions and future directions are presented.

Related Work

Many visualization systems have been developed over the past few decades. These include JavaVis (Oechsle & Schmitt, 2002) which visualizes object and sequence diagrams, one based on WYSIWYC (What You See Is What You Code) model and direct manipulation of program structures called ALVIS LIVE! (Hundhausen & Brown, 2007), and Raptor (Carlisle, Wilson, Humphries, & Hadfield, 2005) a visualization tool that utilizes dataflow diagrams. The main part of the development in this field is focused on algorithm animation, which visualizes data structures and algorithms. Notable algorithm animation tools include JHAVE (Grissom, McNally, & Naps, 2003), BALS-II (Brown, 1988), ZEUS (Brown, 1991), XTANGO (Stasko, 1992), and TRAKLA2 (Malmi et al., 2004).

Boyle, Bradley, Chalk, Jones, and Pickard (2003) paid particular interest in a ‘visual approach’ – portraying the abstract programming concepts with graphical shapes – while defining the new curriculum for London Metropolitan University’s course of introductory programming. Over 600 students took part on the course, and the results of the new ‘blended learning environment’ were quite promising; according to a questionnaire answered at the mid-semester stage more than 80 % of students described their motivation level as high or very high, and over 70 % were happy or very happy about their progress in studies. The increase in pass rates was between 12 and 23 % compared to previous year. Boyle et al. reported some major issues in handling the course transition, but on average they described the graphical approach ‘very successful with the students’ (Boyle et al., 2003, p. 177).

Kannusmäki, Moreno, Myller, and Sutinen (2004) evaluated the use of the Jeliot 3 program visualization system during the second course of programming in the Virtual Studies of Computer Science distance learning program at the University of Joensuu, Finland. The emphasis was on ways of using the tool and on features students would like to have included in the tool. The qualitative data was collected from the course’s discussion forum messages. Gathered data was divided into three categories: usage patterns, usage problems, and opinions and suggestions. Messages in the first category revealed that the students most successful in the course used Jeliot more than the other groups. However, most of the students in general still used other tools to code and test their programs. The usage problems reported were mostly technical or related to the usability of the editor. The animation was criticized on being too slow and some students even found the whole system unnecessary and unsuitable for advanced courses. The positive aspects identified in the feedback included the ability to make conditional statements, loops, and objects more understandable.

Hundhausen, Douglas, and Stasko (2002) conducted a comprehensive meta-study, analyzing 24 experimental studies on effectiveness of algorithm visualization. They state that one of the main reasons visualizations are not widely used is because the teachers responsible for the courses refuse to use new methods in teaching. They also found out that the main focus in articles about visualizations is normally on their graphical means of expression – in other words their visualization capabilities - instead of their learning benefits. Of the 24 studies examined, 11 showed statistically significant results of visualizations positive effects on learning, meaning that the group using a visualization system gained better learning results than the control group. Hundhausen et al. (2002) also discovered that the sole use of visualization systems doesn’t necessarily improve the learning results; it is more important to engage the learners in the subject using visualization system as an aid.

Other studies concerning evaluation of visualization systems include, for example, studies (see Grissom et al., 2003, Laakso, Salakoski, Grandell, et al., 2005; Laakso, Salakoski, Korhonen, 2005) about adapting algorithm animation systems successfully in teaching, and a study about educational impacts of visualization (see Naps et al., 2003). Laakso, Myller, and Korhonen (in

press) studied the effectiveness of algorithm visualization system TRAKLA2 in different engagement levels. With a similar research setup to ours (two hour controlled experiment), they were able to confirm some of the hypotheses presented in the taxonomy of learner engagement with visualization technology (Naps et al., 2002).

VILLE

VILLE is a program visualization tool for teaching programming to novice programmers. Teachers can use the tool in lectures to demonstrate the dynamic behaviour of program execution, and students can use it independently over the web. VILLE contains a predefined set of programming examples grouped into different categories based on their topic. Teachers can easily add new examples to the tool or modify the existing ones. The tool contains also a question editor with which the teacher can attach multiple choice and array related pop-up questions to program events of a chosen programming example. The pop-up questions are then shown to the students as they go through the execution of a programming example, engaging them more deeply in learning process.

VILLE supports all the programming concepts generally featured in introductory programming courses. The support for more advanced concepts is limited: for example objects – excluding arrays, strings and records – are not supported. These limitations however make it possible to define new syntaxes with corresponding features to existing languages in VILLE.

Key Features

This section presents VILLE's key features divided into four categories: level of abstraction, user interaction, tracing execution, and customization. The categories reflect the main functions and features in this tool.

Level of abstraction

Language-independence. One of the most important aspects of VILLE is the ability to view programming examples in several different programming languages. When observing program execution in different languages, a user can discover similarities in their basic functionalities. It is far more important for the novice programmer to learn how different programming concepts actually work than to focus on the syntactical issues of a specific language. We call this the *programming language independency paradigm*.

Defining and adding new languages. As built-in, VILLE supports Java, pseudo code, and C++. The pseudo code's definition can be altered to suit a teacher's needs. It is also possible to define and add new programming languages to further extend the language support.

The parallel view. The program code execution can be viewed simultaneously in two different programming languages. This way the user can see how the execution progresses similarly regardless of syntactical differences between the languages.

Role information. The role information of variables is integrated into the code line explanation. According to Sajaniemi and Kuittinen (2003) the role information of variables helps learning and enhances understanding of programs.

User interaction

Code editing. Besides the example creation and editing view, the program code can also be edited in the visualization view, allowing users to trace the effects of changes in execution and visualization. The user's edits are not saved to the original program.

Pop-up questions. With the built-in editor the teacher can create multiple-choice questions and set them to be triggered at certain stages of the program execution.

Flexible control of the visualization both forwards and backwards. The user can move one step at a time, both forwards and backwards in the execution of a program. Examples can also be run automatically with adjustable speed. Moving backwards in the program execution isn't usually possible in similar applications. Additionally, ViLLE has an execution slider with which the user can progress to any state of the program execution.

Tracing execution

Call stack. The progress of the program execution between different methods due to function calls and returns is visualized with a call stack. When a method is called, a new window is opened on the call stack. The window remains on the stack until the method is finished. When the execution returns to the caller, the return value is shown on top of the stack. The call stack can be especially useful in learning recursion.

Code line explanation. Every code line has an automatically generated explanation, in which all the program events on the line are clearly explained. Furthermore, all possible outputs and variable states are shown. Code line explanation is not a feature in most similar applications.

Visualization row by row. Progress of the program execution is visualized by highlighting rows in the code. In addition to highlighting the program row under execution, ViLLE also highlights the previously executed row with a different colour. This makes the following of the program execution easier.

Breakpoints. The user can set breakpoints in program code lines and move between them both forwards and backwards. This functionality enables debug-based control and observation of the program execution. Backward tracing between breakpoints is not a standard feature in visual debuggers.

Customization

Example collection. ViLLE contains a predefined set of programming examples grouped into categories based on their subject. A user can create new categories and examples or edit the predefined ones. By creating and editing examples, the teacher can illustrate topics essential in his programming courses.

Publish examples. With the export feature ViLLE's examples can be saved to an example collection. The example collection contains a version of ViLLE with example creation and modification functions disabled; runtime modification however is still enabled. The export feature can be used to publish course's programming examples on the web for the students to use.

Visualization View

The visualization view of ViLLE (Figure 1) consists of three areas. The left side of the view contains the program controls and the program code of the current example. The controls can be used to move both forwards and backwards in the visualization. The right side of the view displays the call stack. Each method call creates a new window on top of the call stack, and as the execution of the method is finished, the return value is shown on top of the stack. The fields at the bottom of the view display an explanation of the current program line, program outputs and variable states. The programming example can also be edited in the visualization view to directly see how the modifications affect the execution. Additionally, the call stack area can be replaced with a large variable state visualization area, which visualizes arrays and matrices with graphical presentations.

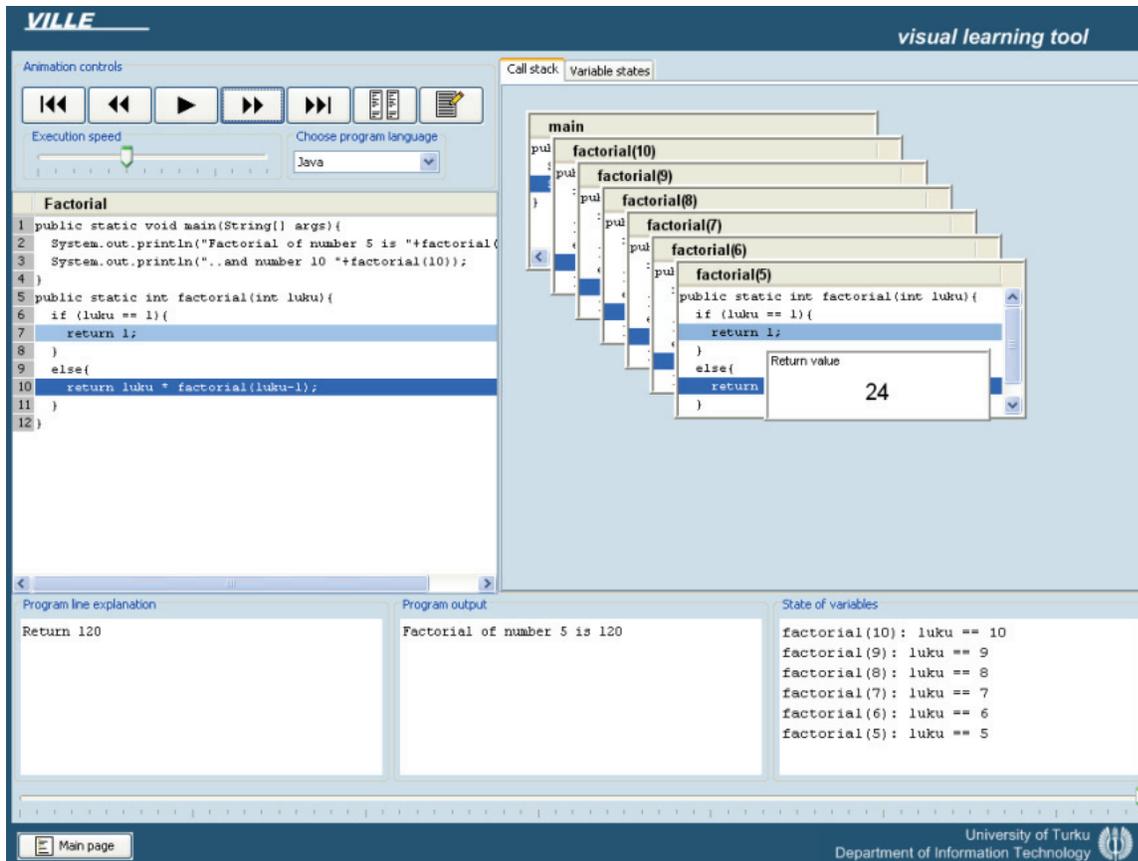


Figure 1: The visualization view in ViLLE

The main idea of ViLLE is to provide a language-independent and, thus, a more abstract view on programming. As built-in, ViLLE supports three programming languages (Java, C++, and a pseudo language) that can be used in the visualization of programs. A user can define a new programming language or modify the existing ones with the built-in syntax editor. This support for multiple languages enables simultaneous viewing of the program visualization in two different languages in parallel, which should help students in understanding the similarity between various programming concepts in imperative programming languages. Another abstraction of programming used in ViLLE is the concept of roles of variables (Sajaniemi, 2002). The role is a description of variable's behaviour in a program. In the visualization view of ViLLE, the program line explanation field also contains information about the roles of variables.

With the above features we try to demonstrate the importance of understanding how the programming concepts actually work in contrast to just learning some specific issues related to the syntax of programming languages. For more detailed information on ViLLE, see Rajala et al. (2007).

Research Design

We conducted an experiment in which we evaluated ViLLE's effectiveness in learning basic programming concepts. There were two main research questions in the study: 1) "Does ViLLE help students in learning to program?", and 2) "Is there any difference in learning when previous programming experience is taken into account?" The null hypotheses were that ViLLE doesn't aid

the learning of basic programming concepts, and the effect is the same for novice and experienced students, respectively.

The experiment was conducted in the third week of the first programming course at the University of Turku. The objective of the course was to learn how computers function logically and to understand the essential concepts of programming. An additional goal was the development of good program reading skills. The course consisted of 28 lecture hours spread over seven weeks. During the first two weeks, topics covered were related to information technology in general instead of programming specifically. Additionally, students had to return four programming assignments at the end of the course.

One two-hour lecture, in which e.g. the syntax of the programming language used was presented, was held before the experiment. A link to ViLLE and its examples was provided to students in the second week and the students were advised to use it before the experimentation. The reason for this was that we wanted them to be familiar with the syntax and the system – including its look and feel. The usage of the tool was however not included in the course's curriculum after the experiment.

The students were divided into two groups: the control group used a textual programming tutorial without access to ViLLE; the treatment group, however, could visualize the examples in the tutorial with the ViLLE tool. The results were analyzed with a two-tailed and pair-wise t-test. In addition, Levene's test was used to calculate the variance for every statistics to determine if the data holds equal or non-equal variances. Unequal variances are marked with '*'-character in presented tables.

Method

The experiment was a between subject design with a pre- and post-test (dependant variable). We had two between-subject factors (independent variables): previous programming experience and previous usage of ViLLE. Students acted alone during the experiment and answered individually to the pre- and post-test. Textual material (provided in a web page) was exactly the same for both conditions, and the only difference was that the treatment group was able to explore integrated examples with ViLLE.

Participants

The participants were university students who attended the first programming course presented in the curriculum. Most of the students were either Computer Science or Mathematics majors.

Students were randomly divided to computer lab sessions and the sessions were randomly assigned to the treatment or the control condition. The total number of participants was 72 ($n = 72$) students. There were 40 students in the control group and 32 students in the treatment group. More than half of the students didn't have any previous knowledge of programming. Moreover, there were three students that participated in the lab session, but who didn't give permission to use their results in this research. There were two lab sessions for each condition. Students who attended the lab session received two bonus points to their final exam results.

Materials

The *pre-test* consisted of three questions. In each question the students were presented a code fragment and asked to define the output or the state of the program. In the first question the program code presented contained three numeric variables and three consecutive conditional statements. The students were supposed to track down the changes in variables and type their values in different points of execution. In the second question the students were presented a loop in

which the values of two variables were changed and printed out. The students were asked to give the complete output of the program. The third question included a recursive function which calculated the sum of the sequence from given parameter down to 1. The assumption was that the students with some earlier programming experience would be able to solve at least some of the assignments. In addition, the students were asked some background information, including the amount of earlier programming experience on the scale of 0 to 4, programming languages they had used, and whether they had used ViLLE before taking the test.

After completing the pre-test the students went through a *programming tutorial* that we had prepared earlier. The tutorial consisted of few basic programming subjects – the same subjects the students were tested on with pre- and post-tests. The subjects covered (in this order) variable usage and manipulation, printing, conditional statements, loop statement (*while*-statement, to be exact), function calls, and finally recursive functions. There was a textual description on all topics with some examples on how to use them. The tutorial contained 14 programming examples, and the students were instructed to write down the output of each example on paper. This was to ensure that each student really went through the tutorial. The group using ViLLE could examine the execution of each example by selecting a link titled ‘*run this example*’ next to it.

The *post-test* included all the questions of the pre-test in exactly the same form. In addition, there were two extra questions. In the first one the students were asked to complete the given program code so that it would output all the even numbers from 2 to 24. The template given consisted of *while* and *print*-statements without parameters and some blank fields with proper indentations for the students to fill in. The second question was a follow-up to the question about the recursive function: the students were asked to deduce the outcome of the same function with two different parameters.

Procedure

The study was performed in the third week of the seven week course at the computer lab sessions that lasted for two hours. The students were divided randomly into two groups. Both groups had the same programming tutorial, but the second group could additionally execute the examples in the tutorial with ViLLE. In the beginning of the session students took the pre-test independently. The time reserved for filling out the questions was 15 minutes.

After the pre-test each of the students used the programming tutorial for 45 minutes. To monitor the involvement, the students were instructed to write down the output of each example (14 in total). Students went through the tutorial independently; they were allowed to ask for assistance only if they encountered technical difficulties.

The session ended with answering the post-test. Since the post-test had two extra questions compared to the pre-test (and since the extra questions were more demanding) the time reserved for answering the questions was 30 minutes.

Each question in the pre- and post-tests was analyzed in the scale of 0 to 10. Zero points meant that the answer was totally wrong, and each point advanced meant the increase of 10 percent in the correctness of the answer. The total maximum in the pre-test was 30 points and in the post-test 50 points.

Results

Effectiveness of ViLLE

In this section we present results to research question related to the independent variable of using ViLLE. The treatment group used ViLLE in the lab session while the control group didn't use it

at all. There were 32 students in the treatment group and 40 students in the control group. The groups were randomly formed.

Previous knowledge

Table 1 presents the results from pre-test for the treatment and the control group. The table includes averages, standard deviations (in parenthesis) and p-values obtained from two-tailed t-test.

Table 1: Pre-test results

Question	Control (n = 40)	Treatment (n = 32)	p-value
Question 1 (Q1)	5.20 (2.67)	6.19 (2.46)	0.111
Question 2 (Q2)	2.70 (3.53)	2.13 (3.53)	0.494
Question 3 (Q3)	2.68 (4.15)	2.09 (3.88)	0.546
Total	10.58 (8.64)	10.41 (7.18)	0.930

There were no statistically significant differences between groups in any pre-test questions. In absolute scale, the control group outperformed treatment group in Q2 and Q3, while the treatment group achieved more points in Q1.

As stated earlier, students were advised to familiarize themselves with ViLLE's interface before the test; 20 students in the control group and 19 students in the treatment group reported having done this. There were no statistically significant differences inside or between the groups in pre-test results related to ViLLE's previous usage.

Previous programming experience

We also asked about students' previous programming experience and divided the treatment group and the control group based on this covariant (previous programming experience). The question's scale was from 0 to 4. Based on this gathered data we computed a new discrete (boolean) 0,1-variable for previous programming experience; 0 is equal to no previous programming experience (NPE) and all the other values were counted for some previous experience (SPE). Tables 2 and 3 present the pre-test results between following groups: 1) treatment and NPE vs. control and NPE 2) treatment and SPE vs. control and SPE, respectively.

Table 2: Pre-test results of students with no previous programming experience (NPE)

Question	Control (n = 23)	Treatment (n = 20)	p-value
Q1	4.17 (2.39)	5.60 (2.33)	0.041
Q2	1.22 (1.78)	1.00 (2.22)	0.724
Q3	1.00 (2.86)	1.65 (3.62)	0.514
Total	6.39 (4.68)	8.25 (5.44)	0.235

Table 3: Pre-test results of students with some previous programming experience (SPE)

Question	Control (n = 17)	Treatment (n = 12)	p-value
Q1	6.59 (2.53)	7.17 (2.76)	0.564
Q2	4.71 (4.31)	4.00 (4.51)	0.673
Q3	4.94 (4.62)	2.83 (4.36)	0.226
Total	16.24 (9.63)	14.00 (8.48)	0.524

There were no statistically significant differences between the treatment and control groups. Notice that in Q1 in Table 2, the seemingly significant p-value (0,041) does not indicate a statistically significant difference, because there were three questions in the pre-test and thus the p-value should be three times smaller (Bonferroni correction). Based on the data from the Tables 1, 2, and 3, we conclude that there is no difference between the control and the treatment group while comparing the pre-test data with or without the previous experience of programming.

Post-test results

The post-test included all the questions presented in the pre-test, with two additional questions. Table 4 presents statistics for the control group and the treatment group. In the first column (question) there is a correspondent pre-test question label. The table includes averages, standard deviations (in parenthesis) and p-values obtained from two-tailed t-test for each question. In addition, there are total points of shared questions (pre- and post-test), total points (post-test), differences between each question in the pre- and post-test and total difference.

We also calculated Cronbach's alpha reliability values for pre- and post-test questions. The results (pre-test $\alpha = 0,667$ and post-test $\alpha = 0,831$) indicate high reliability.

Table 4: Post-test results

Question	Control (n = 40)	Treatment (n = 32)	p-value
PQ1 (Q1)	6.30 (2.81)	6.13 (2.69)	0.790
PQ2 (Q2)	5.10 (4.35)	5.50 (4.50)	0.704
PQ3	6.28 (3.75)	5.88 (3.75)	0.654
PQ4 (Q3)	6.15 (4.56)	6.50 (4.42)	0.744
PQ5	7.05 (3.78)	6.69 (4.08)	0.698
Total (shared)	17.55 (9.08)	18.13 (8.81)	0.788
Total (all)	30.88 (15.20)	30.69 (15.08)	0.959
Diff PQ1	1.10 (2.60)	-0.06 (2.81)	0.073
Diff PQ2	2.40 (3.30)	3.38 (4.02)	0.262
Diff PQ4	3.48 (4.81)	4.41 (4.53)	0.405
Total diff	6.98 (6.81)	7.72 (6.76)	0.646

When comparing the shared questions in the pre- and post-test, we see that in absolute scale the control group outperformed the treatment group in PQ1 while the treatment group did better in PQ2 and PQ3. The better performance in PQ1 is related to the fact that achieved points were quite high in treatment group in pre-test. Still, the differences are too small to reject the null hypothesis. Similarly to the pre-test, the previous usage of ViLLE as a factor didn't reveal any statistically significant differences, either inside or between the groups.

The same statistics calculated with the previous programming experience taken into account are shown in Tables 5 and 6. (*-character indicates non-equal variances)

Table 5: Post-test results with NPE

Question	Control (n = 23)	Treatment (n = 20)	p-value
PQ1 (Q1)	5.74 (2.78)	5.90 (2.86)	0.853
PQ2 (Q2)	3.39 (3.97)	4.70 (4.58)	0.321
PQ3	5.30 (4.06)	5.05 (3.65)	0.831
PQ4 (Q3)	5.22 (4.83)	6.00 (4.71)	0.595
PQ5	6.09 (4.09)	6.05 (4.20)	0.977
Total (shared)	14.35 (8.27)	16.60 (9.29)	0.405
Total (all)	25.74 (14.44)	27.70 (15.49)	0.670
Diff PQ1	1.57 (2.48)	0.30 (2.62)	0.113*
Diff PQ2	2.17 (3.07)	3.70 (4.38)	0.189
Diff PQ4	4.22 (4.73)	4.35 (4.73)	0.927
Total diff	7.96 (5.80)	8.35 (7.98)	0.853

Table 6: Post-test results with SPE

Question	Control (n = 17)	Treatment (n = 12)	p-value
PQ1 (Q1)	7.06 (2.75)	6.50 (2.43)	0.577
PQ2 (Q2)	7.41 (3.81)	6.83 (4.22)	0.703
PQ3	7.59 (2.90)	7.25 (3.65)	0.783
PQ4 (Q3)	7.41 (3.94)	7.33 (3.94)	0.958*
PQ5	8.35 (2.96)	7.75 (3.82)	0.635
Total (shared)	21.88 (8.51)	20.67 (7.64)	0.696
Total (all)	37.82 (13.68)	35.67 (13.53)	0.678
Diff PQ1	0.47 (2.70)	-0.67 (3.11)	0.303
Diff PQ2	2.71 (3.65)	2.83 (3.46)	0.925
Diff PQ4	2.47 (4.87)	4.50 (4.38)	0.252*
Total diff	5.65 (7.98)	6.67 (4.14)	0.689

As seen in Tables 5 and 6, the previous programming experience had no statistically significant effect. The previous statistics are summarized in Table 7, including the averages from the pre-test, post-test, differences and p-values for the treatment group, the control group, treatment with NPE (TNPE), treatment with SPE (TSPE), control with NPE (CNPE), and control with SPE (CSPE). The p-value is obtained by comparing total points from the pre- and post-test in shared questions with a pair-wise t-test.

Table 7: Pre- and post-test results

Points	Control (C)	Treatment (T)	CNPE	CSPE	TNPE	TSPE
Pre-test	10.58	10.41	6.39	16.24	8.25	14.00
Post-test	17.55	18.13	14.35	21.88	16.60	20.67
Total diff	6.98	7.72	7.96	5.65	8.35	6.67
p-value	0.000	0.000	0.000	0.010	0.000	0.000

Statistics in the table 7 confirm that learning occurred in both groups, and there was a statistically very significant difference between pre- and post-test results ($p \leq 0.01$) in all groups.

Based on the data presented, we can not fully reject our null hypothesis, which was that ViLLE does not aid the learning of basic programming concepts. The absolute values and the difference between CSPE and TSPE groups, however, indicate that there is a trend towards treatment group, suggesting that ViLLE might have a positive effect on students' learning.

Novices vs. Experienced

The other research question was, whether the effect of ViLLE is the same for novices and experienced students. The null hypothesis was that there is no difference between novices and experienced students. The treatment and control groups were both divided into two groups based on the previous programming experience. In contrast to the first research question, the students' results were compared *inside* the group, rather than between the groups.

Previous knowledge

The results from the pre-test are compared between novices (NPE) and experienced (SPE) in the control group (Table 8) and the treatment group (Table 9). Tables include averages, standard deviations (in parenthesis) and p-values obtained from two-tailed t-test for each question separately and for total number of points acquired in the pre-test.

Table 8: Pre-test scores for CNPE and CSPE

Question	Control and NPE (n = 20)	Control and SPE (n = 12)	p-value
Q1	4.17 (2.33)	6.59 (2.53)	0.003
Q2	1.22 (1.78)	4.71 (4.31)	0.005*
Q3	1.00 (2.86)	4.94 (4.62)	0.005
Total	6.39 (4.68)	16.24 (9.63)	0.001*

Table 9: Pre-test scores for TNPE and TSPE

Question	Treatment and NPE (n = 20)	Treatment and SPE (n = 12)	p-value
Q1	5.60 (2.11)	7.17 (2.76)	0.107*
Q2	1.00 (2.22)	4.00 (4.51)	0.049*
Q3	1.65 (3.62)	2.83 (4.34)	0.414
Total	8.25 (5.44)	14.00 (8.48)	0.051*

We can see that there is a statistically very significant difference between CNPE and CSPE. The difference between TNPE and TSPE is also statistically significant ($t(30) = -2.11, p = 0.051$). Thus, we conclude that there is statistically significant difference between NPE and SPE in both groups.

Post-test results

Table 10 presents statistics between CNPE and CSPE and Table 11 between TNPE and TSPE. In the first column the correspondent pre-test question label is shown in parenthesis. The tables include averages, standard deviations (in parenthesis), and p-values obtained from two-tailed t-test for each question. In addition, the total points of shared questions (pre- and post-test), total points (post-test), differences between each question in the pre- and post-test, and the total difference in shared questions are displayed.

Table 10: Post-test scores for CNPE and CSPE

Question	CNPE (n = 23)	CSPE (n = 17)	p-value
PQ1 (Q1)	5.74 (2.78)	7.06 (2.75)	0.144
PQ2 (Q2)	3.39 (3.97)	7.41 (3.81)	0.003
PQ3	5.30 (4.06)	7.59 (2.90)	0.045*
PQ4 (Q3)	5.22 (4.83)	7.41 (3.94)	0.122*
PQ5	6.09 (4.09)	8.35 (2.96)	0.049*
Total (shared)	14.35 (8.27)	21.88 (8.51)	0.008
Total (all)	25.74 (14.44)	37.82 (13.68)	0.011
Diff PQ1	1.57 (2.48)	0.47 (2.70)	0.198*
Diff PQ2	2.17 (3.07)	2.71 (3.65)	0.620
Diff PQ4	4.22 (4.73)	2.47 (4.87)	0.261
Total diff	7.96 (5.80)	5.65 (7.98)	0.295

Table 10 shows that there is a statistically very significant difference between CNPE and CSPE in the post-test scores. The same phenomenon was observed also in the pre-test. As shown in Table 7, learning occurred both in CNPE ($p < 0.01$) and CSPE ($p < 0.05$). Yet, a very significant difference remains between CNPE and CSPE in shared questions ($p = 0.008$), and there also is a very significant difference ($p = 0.011$) in the total points in the post-test.

Table 11: Post-test scores for TNPE and TSPE

Question	TNPE (n = 20)	TSPE (n = 12)	p-value
PQ1 (Q1)	5.90 (2.86)	6.50 (2.43)	0.533*
PQ2 (Q2)	4.70 (4.58)	6.83 (4.22)	0.199
PQ3	5.05 (3.65)	7.25 (3.65)	0.109
PQ4 (Q3)	6.00 (4.71)	7.33 (3.94)	0.418
PQ5	6.05 (4.20)	7.75 (3.82)	0.261
Total (shared)	16.60 (9.29)	20.67 (7.64)	0.212
Total (all)	27.70 (15.49)	35.67 (13.53)	0.151
Diff PQ1	0.30 (2.62)	-0.67 (3.11)	0.354
Diff PQ2	3.70 (4.38)	2.83 (3.46)	0.564
Diff PQ4	4.35 (4.73)	4.50 (4.38)	0.929
Total diff	8.35 (7.98)	6.67 (4.14)	0.439*

As seen in Table 9, the difference between TNPE and TSPE in the pre-test was statistically significant ($p = 0.051$). In the post-test, however, there was no statistically significant difference in any of the questions, in total points, or in differences in the shared questions (see Table 11).

Therefore, the null hypothesis can be rejected, and we can conclude that ViLLE is more beneficial for the novice students than for the experienced ones.

Discussion

The evaluation of our research results was studied in two separate cases. In the first case we compared the learning results of the treatment and the control group. The control group used only a textual programming tutorial, while the treatment group using the same material could in addition execute the examples with ViLLE.

In the first research question we compared learning results between control and treatment groups. We found no statistically significant difference between the groups, and thus we can not reject the null hypothesis. Similarly we found no difference in results between genders, between students' that had used ViLLE before the research, or students' with no earlier experience with ViLLE.

In absolute scale, the results favoured the treatment group, indicating that ViLLE might have a positive effect on students' learning. However, the differences were too small in order to get statistically significant results. One reason for that might be that the treatment group's students were coping with a heavier cognitive load (see Chandler & Sweller, 1996) due to the fact that they used ViLLE in addition to the textual material. This load was even heavier for those who hadn't used ViLLE beforehand. We believe that the cognitive load combined with the short learning session was the primary reason for not achieving statistically significant results between treatment and control groups. Another reason might be the low count of participants ($n=32$ in treatment group; $n=40$ in control group) as well as the short duration of the experiment. However, there was a significant (at 0.01 level) medium correlation (0.452) between the post-test and the final exam scores. Hence, the results of the two hour session seem to somewhat predict the outcome of the whole course.

In the second research question, we compared the students' learning performance inside both groups when previous programming experience was taken into account. The treatment group was divided into two groups: one with no previous programming experience (TNPE) and the other with some previous programming experience (TSPE). In the pre-test, the difference between the groups was statistically significant ($p = 0.051$). In the post-test, on the other hand, there was no statistically significant difference at all. So, there is solid evidence that ViLLE is more beneficial to novices, and thus we can reject the null hypothesis. The control group was divided identically to CNPE and CSPE. There was a statistically very significant difference between these groups both in the pre- and post-test, which was opposite to TNPE vs. TSPE. Hence, it seems that ViLLE has a substantial effect on narrowing the gap between novices and more experienced students. The learners' short exposure to the tool makes the result even more remarkable.

With these findings combined, it seems that ViLLE enhances students' learning of basic programming concepts. ViLLE proved to be particularly beneficial for novice students, effectively evening out the differences caused by previous programming experience.

Conclusions

We conducted an experiment focusing on program visualization's effectiveness on learning basic programming concepts. We utilized the ViLLE tool in the first programming course at the University of Turku. We found evidence that program visualization, more specifically the ViLLE tool, enhances students' learning regardless of previous programming experience. Moreover, it seems that the tool benefits novice learners more than learners with some previous experience. The differences between the novices and more experienced learners disappeared in the treatment group during a very short training period. In the future, we plan to carry out a study in which ViLLE is used throughout the course and evaluate its individual features separately.

Acknowledgments

This work was supported by the Academy of Finland under grant number 111396.

References

- Ben-Ari, M. (2001). Program visualization in theory and practice. *Informatik/Informatique*, 2, 8-11.
- Boyle, T., Bradley, C., Chalk, P., Jones, R. & Pickard, P. (2003). Using blended learning to improve student success rates in learning to program. *Journal of Educational Media, special edition on Blended Learning*, 28(2-3), 165-178.
- Brown, M. H. (1988). Exploring algorithms using Balsa II. *IEEE Computer*, 21(5), 14-36.
- Brown, M. H. (1991). Zeus: A system for algorithm animation and multi-view editing. *Proceedings of IEEE Workshop on Visual Languages*, 4-9. New York: IEEE Computer Society Press.
- Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M. (2005). RAPTOR: A visual programming environment for teaching algorithmic problem solving. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, St. Louis, Missouri, USA, 176-180.
- Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychol*, 10, 151-170.
- Grissom, S., McNally, M., & Naps, T. (2003). Algorithm visualization in CS education: Comparing levels of student engagement. *Proceedings of the ACM Symposium on Software Visualization*, San Diego, California, 87-94.
- Hundhausen, C. D., & Brown, J. L. (2007). What you see is what you code: A 'live' algorithm development and visualization environment for novice learners. *Journal of Visual Languages and Computing*, 18(1), 22-47.

Effectiveness of Program Visualization

- Hundhausen, C. D., Douglas, S. A. & Stasko, J. D. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13, 259-290.
- Hyrskykari, A. (1993). Development of program visualization systems. Report, Department of Computer Science, University of Tampere, Finland. Presented at the *2nd Czech British Symposium of Visual Aspects of Man-Machine Systems*, Praha, 1-21.
- Kannusmäki, O., Moreno, A., Myller, N., & Sutinen, E. (2004). What a novice wants: Students using program visualization in distance programming course. *Proceedings of the Third Program Visualization Workshop (PVW'04)*, Warwick, UK.
- Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology*, 13(4).
- Laakso, M.-J., Myller, N., & Korhonen, A. (in press). Comparing learning performance of students using algorithm visualizations collaboratively on different engagement levels. *Journal of Educational Technology and Society*.
- Laakso, M.-J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A., & Malmi, L. (2005). Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1), 49-68.
- Laakso, M.-J., Salakoski, T., & Korhonen, A. (2005). The feasibility of automatic assessment and feedback. *Proceedings of Cognition and Exploratory Learning in Digital Age (CELDA 2005)*. IEEE Technical Committee on Learning Technology and Japanese Society of Information and Systems in Education, Porto, Portugal, 113-122.
- Lister, R., Adams, S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. (2004). A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin*, 36(4), 119-150.
- Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., & Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2), 267-288.
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., et al. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4), 125-140.
- Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. *Proceedings of the Working Conference on Advanced Visual Interfaces*, Gallipoli, Italy, 373-376.
- Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., et al. (2003). Evaluating the educational impact of visualization. *Working group reports from ITiCSE on Innovation and Technology in Computer Science Education*, ACM Press, 124-136.
- Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., et al. (2002). Exploring the role of visualization and engagement in computer science education. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, 35, 2, 131-152.
- Nikula, U., Sajaniemi, J., Tedre, M. & Wray, S. (2007). Python and Roles of Variables in Introductory Programming: Experiences from Three Educational Institutions. *The Journal of Information Technology Education*, 6, 199-214. Retrieved from <http://jite.org/documents/Vol6/JIIEv6p199-214Nikula269.pdf>
- Oechsle, R. & Schmitt, T. (2002). JAVAVIS: Automatic program visualization with object and sequence diagrams using the java debug interface (JDI). *Lecture Notes in Computer Science, Vol. 2269: Software Visualization*, 176-190.
- Rajala, T., Laakso, M.-J., Kaila, E. & Salakoski, T. (2007). VILLE – A language-independent program visualization tool. *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007)*, Koli National Park, Finland, November 15-18, 2007. *Conferences in Research and Practice in Information Technology, Vol. 88*, Australian Computer Society. Raymond Lister and Simon, Eds.

- Sajaniemi, J. (2002). PlanAni – A system for visualizing roles of variables to novice programmers. *University of Joensuu, Department of Computer Science, Technical Report, Series A, Report A-2002-4*.
- Sajaniemi, J. & Kuitinen, M. (2003). Program animation based on the roles of variables. *Proceedings of the 2003 ACM Symposium on Software Visualization*, San Diego, California, 7-ff
- Stasko, J. (1992). Animating algorithms with XTANGO. *ACM SIGACT News*, 23(2), 67-71.
- Tenenberg, J., Fincher, S., Blaha, K., Bouvier, D., Chen, T.-Y., Chinn, D., et al. (2005). Students designing software: A multi-national, multi-institutional study. *Informatics in Education*, 4(1), 143-162.

Biographies



Teemu Rajala is a PhD student at University of Turku. He received his master's degree from the same university in 2007. His research focuses on visualization of programs and algorithmic problem solving.



Mikko-Jussi Laakso is a PhD student working as a researcher in a joint project of University of Turku and Helsinki University of Technology. He received his M.Sc (Computer Science) in 2003. His research interest covers program and algorithm visualization, learning environments, computer aided and automatic assessment in computer science education.



Erkki Kaila has written his Master's thesis on program visualization in programming learning in University of Turku. His research interests include program visualization systems and IT education.



Tapio Salakoski is a professor of Computer Science at University of Turku, where he received his Ph.D. in 1997. His main research focus has been in methodology development using machine learning and other intelligent techniques. He is leading a multidisciplinary research group studying various task domains, including problems related to human learning and computing education research.