How a Visualization Tool Can Be Used - Evaluating a Tool in a Research & Development Project

Matti Lattu [1 Jorma Tarhio [2 Veijo Meisalo [1

1) Department of Teacher Education
2) Department of Computer Science
University of Helsinki
firstname.lastname@helsinki.fi

Keywords: POP-III.B. java; POP-III.B. visualisation; POP-V.B. case studies

Abstract

This paper outlines a part of a larger qualitative evaluation study where Jeliot, a tool designed to aid students in understanding algorithms, was used in a real classroom situation by two different groups. According to the findings, the tool could be used in an introductory programming course and students found visualization helpful. However, the making SV automatic is not a straightforward task, as the tool should be able to understand the relations of the variables. Based on the findings we suggest some further development areas for Jeliot.

Introduction

Evaluation of new learning environments is important when developing them. This study reports an evaluation of Jeliot (see eg. Haajanen et al. 1997), a Web-based animation environment1 for learning algorithms and data structures. The evaluation was conducted by a researcher independent of the group that developed Jeliot.

Jeliot animates algorithms (or programs) written in the Java programming language by visualizing data structures as smoothly moving graphical objects. Jeliot aims at automatic animation, i.e. the user submits a plain algorithm and Jeliot generates and shows its animation. The design and the operational framework of Jeliot are similar to those of Eliot (Lahtinen, Sutinen & Tarhio 1998), an earlier visualizing tool developed by the same group. Eliot was based on the C language and worked in the X windows environment. Jeliot (short for Java-Eliot), written in Java, follows the visualization model of Eliot in the World-Wide Web environment, although its architecture is completely different.

The usability of Eliot in teaching has been evaluated by its development group, and these results apply directly to Jeliot. Based on Eliot, an alternate teaching scheme for a data structure laboratory course, applying problem solving, was created (Meisalo, Sutinen & Tarhio 1997). Two studies were conducted to evaluate the influence of Eliot in learning data structures in a laboratory course. The first study showed that using Eliot improved the motivation and activation level of the participating students (Meisalo et al. 1997). The second study indicated that using Eliot the students produced higher quality code and documentation, as well (Markkanen et al. 1998).

The authors are aware of two further, rather small scale studies on Jeliot. The first one dealt with cross-cultural co-operation in teaching programming (Järvinen et al. 1999). In the other case study a small group of high school students became acquainted with the properties of Jeliot during their course of Modern Information and Communication Technologies.

[1] http://www.cs.helsinki.fi/research/aaps/Jeliot/

In the present study, the emphasis is on testing the applicability of Jeliot in an introductory programming course at university level and in courses for high school students. The leading pedagogical principle in using a tool like Jeliot is to visualize abstract algorithms and help in localizing possible errors in the logic or implementation of a program.

Jeliot

To understand the present paper, the reader should know how Jeliot is used and something about its features. Jeliot is based on a theater metaphor (Lahtinen, Sutinen & Tarhio 1998). The algorithm or program given by the user is a script of a play where the variables are acting the roles. The user is the director of the play selecting the visual appearance of the variables. The theater metaphor was also seen useful by McWhirter (1996), who proposed that the student should not be a producer but an actor. By that he meant that the creating of the visualization should not be the job of the student but something that the visualization system would do. This is exactly the idea behind Jeliot.

Jeliot is used with a Java-capable Web browser. The user writes a Java program as a local file and submits it to the Jeliot Web page which forwards it to the Jeliot server to be compiled. The server returns either an executable program or a brief error message. After a successful compilation, the director (the user) selects the actors (the variables to be visualized) and their costumes (the visual representations of the variables) and orders the play to be started either continuously or line by line.

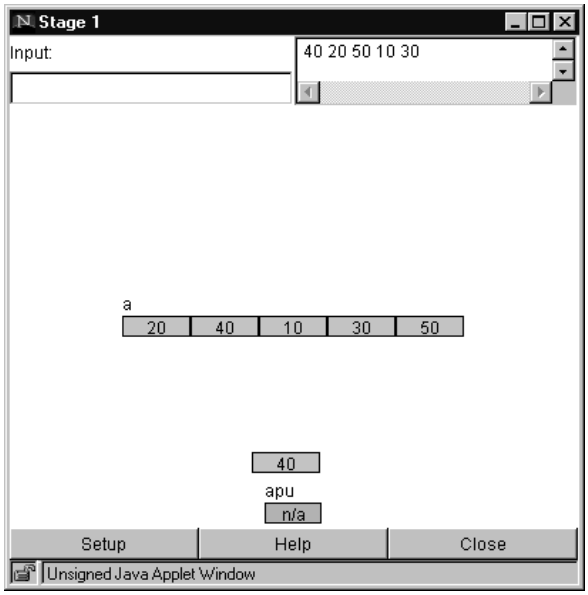

Figure 1. A snapshot of the Stage Window during a sorting algorithm. The array is in the middle. A box including value "40" is on its way to the temporary array below.

Jeliot supports gradual solving processes. Based on the observations on the evolving animation, the user modifies his program to improve its logic, submits the modified code, and repeats the cycle described above as many times as needed.

Technically, animation is controlled by the operations of data types. If the program uses the animated data types provided by Jeliot, animation will be automatic because animation has been embedded in the implementation of data type operations. Currently Jeliot is capable of animating all the primitive types of Java (boolean, integral, and floating-point types), one and two dimensional arrays, stack and queue, all with a selection of visual representations.

The Research on Software Visualization

The most studies dealing with SV are focusing on the efficiency of the visualized program vs. traditional methods in teaching. Stasko and Lawrence (1998) argue for example, that the key of the research should be: “Do algorithm animations truly assist teaching and learning, and if so, how can they be best utilized?” (see also Byrne, Catrambone & Stasko 1999).

The question is an important one. It is essential in science that new ideas are first tested and the assessment is based on the results. Stasko and his colleagues approach the SV with the psychologist's viewpoint by stressing conceptualization and memorizing. Of course, these are the fundamental questions in learning and the research in this area is most valuable.

Learning, as understood in the psychology, is not the only thing that is interesting in CS education. Motivation, existing teaching and learning practices, the nature of the content to learn, and the resources available, just to mention a few, play an important role in the everyday teaching and learning. The approach adopted by us sees the learning and teaching in the context of the classroom, which is a marketplace of various practices and intentions.

In the animation tools developed and studied by the Stasko's group, namely Polka, XTango and Samba, the basic idea is that the user is creating the animations. The user's activity is directed towards designing the visual appearance of the algorithm. As Jeliot creates animations automatically, the user's activity is focusing to the actual algorithm coded to a Java program. In a nutcase, while Stasko's approach stresses the animation as a part of a learning process, we see the visualization created by Jeliot as a part of a programming activity.

After understanding this difference, it is easier to understand the difference in the selected research strategies. It is evident that we need both kinds of research. A firm psychological basic theory about relations between memory, the development of concepts, and perception (in the case of SV, visualization and animation) would have an enormous impact on teaching and learning. While waiting for such a theory, we should focus on the various ways SV can be applied to support the existing teaching and learning methods and on the possibility that it could promote totally new practices. In these studies, the questions should be broad and there should always be a possibility to focus on emerging interesting phenomena.

Algorithm animation could be one way to help the students or even laymen to understand the logic of algorithms. Visualization could also encourage the students to approach debugging in a more analytic way, as it makes the programs and their execution more transparent than the traditional programming environments. Using a visualization tool in teamwork situations was shown to be beneficial in an earlier experiment (Meisalo, Sutinen & Tarhio 1997).

Empirical Study and Methodology

This paper reports a smaller part of the larger research which was carried out in the fall term 1998. We will here give an outline of an answer to one of the research questions: What kind of experiences do we get when using Jeliot? Based on the results we expect to be able to detect further development needs in our research and development project, and share these with persons working with similar domains.

Because we expected Jeliot to be widely applicable in different environments, we decided to make the experiment in two different groups. Also, a large variety of feedback was expected to get: new technical problems, users' feelings (both teachers' and students'), and information of various ways the Jeliot could be used. The resources available would not cover running a reliable quasi-experimental study with respective control groups. Also, as explained before, running such an experiment would not have been rewarding at this stage.

Studyt Group 1: CS Students

The first group in the study consisted of three assignment groups on a university level introductory programming course. The course covers basic principles of programming and Java: algorithms, error detection and correction, and object orientation.

The course consists of 52 hours of lectures and 24 hours of assignment sessions. 564 students entered for the course, 493 (87% of those who entered) did attend at least one assignment session and only 268 (48% of those who entered) passed the course. One reason for the high drop-out percent is that there are no restrictions or fees for the students from other faculties to attend the basic courses. In the experimental assignment groups there were 50-80 students altogether.

The teacher of the experimental assignment groups was a M.Sc. with several years experience in teaching CS. The lectures and the assignments were provided by the lecturer. No formal training of the use of Jeliot was arranged for Jeliot, as controls resemble an ordinary debugging utility. According to the initial plan the students would have got examples of the strategies of use through the assignment session demos.

In Group 1 Jeliot was not widely utilized as it lacked general-purpose input and output routines. In the students' interviews after the course it turned out that most of the interviewees had tried Jeliot once or twice but lack of guidance and lack of the students' interest had prevented regular use.

Study Group 2: High School Pupils

To get wider experience of the use of Jeliot, a 10-12 hour Java programming course was offered to the high schools of the local municipality. Three schools were interested to combine the course with their regular teaching and 37 pupils attended the courses. Two of the courses were based on computer laboratory teaching, using short lectures, independent learning as well as teamwork (Groups 2a and 2b, 20 pupils). The third group was run as a distance education course (Group 2c, 9 students).

During the course, the pupils were introduced to the basic structures in programming: variables, loop structures and subprograms.

In Group 2 the starting point of all planning was to use Jeliot as much as possible. The assignments were prepared in concordance with the nature of Jeliot. Therefore, there were not the same difficulties as in Group 1, where Jeliot was lacking features needed in the assignments. Making observations of the learning process was possible in Groups 2a and 2b. In these groups the focus of the data collection was directed at the teacher's and pupils' use of Jeliot, and at how the visualization of variables seemed to help explanation and learning. Also possible problems and disadvantages in the use of Jeliot were to be recorded.

Data Collection

The study was carried out using the qualitative approach and it could be categorized as an evaluation study.

Students in Group 1 were interviewed both before (38 students) and after (11 students) the course while the Group 2 pupils were interviewed only after the course (23 students) because their course was considerably shorter than in Group 1. Interviews were semi-structured and they were carried out in a separate room depending on the facilities offered by the field situations. The recordings (typically about 15 minutes) were transcribed by the interviewer for further analysis. We did not want to use questionnaires since asking clarifying questions had not been possible.

The interview data used in this paper is mostly from the interviews held after the Group 1 and 2 courses. The interviewed students in Group 1 (n=11) had volunteered for the first interview and then selected by matching to the latter one. Statistically, their average did not differ significantly from the mean grade of the rest of the course. The fact that only one failed student was interviewed after the course does not distort the results as the paper is not dealing with motivation or course-goal related matters.

A non-structured observation was carried out during the lectures. The independent researcher followed the session making notes. Afterwards he typed up the notes. The typing was usually done immediately after the observation to increase the credibility of the data (Lincoln & Guba 1985).

Data Analysis

Both the transcribed interviews, field notes, and some email conversations between the teachers and the Jeliot developers were transcribed with computer and imported to qualitative data analysis program Aquad 5. This database included 126 separate documents.

The analysis began by reading the material and making some low-level coding. Based on this exploration the final focus areas were selected, and final categories were chosen. The low-level codes were further processed in a sense of the open coding phase of the grounded theory presented by Strauss and Corbin (1991). The aim was to create groups which would cover similar text units and present different angles and views to this massive data.

The categories were defined as overlapping because of the nature of the data. In some rare cases, the findings clearly belonged to more than one category. However, as the categories still had their distinctive properties, they could not be merged.

After coding, the categorization was verified by another researcher.

Findings

In the beginning of the field period, one of the interests was to collect the experiences of using Jeliot. First the use of Jeliot is described followed by the helpful aspects and the shortcomings.

How Was Jeliot Utilized?

When talking about a new innovation for teaching and learning, the essential question should be how these tools are utilized and what new possibilities they are offering.

A traditional way to utilize a visualization tool is to use it *to support teacher's oral presentations*. In this experiment, this was also observed. The teachers tended to use the continuous execution mode instead of running the program step by step and that way gaining more time for presentation. To gain time for explanations the teacher turned the animation speed down.

Students also used the continuous play mode when presenting their programs. The step-by-step mode was applied only when it was recommended by the teacher. In Group 2 an interesting example of a pupil's presentation was observed. Pupil A had completed an exercise and two other pupils wanted to see the program. Asked by the observer, pupil A showed the visualization to the others but hid the program code. This strategy could be useful in helping novice programmers in restructuring their cognitive images to better suit the logic of the computer. When the student has created an idea of a possible solution but is unable to formulate an algorithm, taking a look at a few visualizations related to the problem could help in constructing the code (see tutoring below).

A typical way to teach new language features in the courses of Group 2a and 2b was *to give an example program* to the pupils. The teacher was using two different strategies: either he started with his own presentation and continued with pupil activity, or the activity proceeded the summarizing debriefing. In most cases the pupils were asked to modify the program and examine the differences in the execution. The teacher was relying heavily on the visualization of Jeliot. To name an example, when introducing the array variables the teacher did not present any illustration to explain the concept but presented the language syntax and an example program.

There is nothing new in using sample programs and modifying them, but visualization offers a way to study algorithms without understanding the actual code behind it. In the courses of Groups 2a and 2b the pupils studied the visualizations of different sorting algorithms. This was done in groups of 2-3 pupils. After the pupils had studied the animations for 5-10 minutes they explained their sorting

method to others with the help of Jeliot. The explanations and the ensuing conversations showed that anyhow many of the students had understood the operation of their sorting algorithm.

Tutoring is an important part of lab work. During the Group 2 lab sessions, the pupils did a lot of independent exercises using Jeliot. Notes from the observations indicate that Jeliot was usable in tutoring, as soon as the pupils had a code to visualize.

However, the main problem in supporting the problem solving of a novice programmer is in finding a way to help the students in transforming their cognitive ideas into the form of an algorithm:

> "You find yourself stumbling, when you're used to human language, you stumble when the computer doesn't understand... simple ones. When you have to say some mathematical formulas in a completely different way. That, I mean that way of thinking, that makes it difficult." (Group 2)

For such occasions, the good old paper-and-pen was the only tool that could be used.

In Group 2 courses the pupils did a lot of *programming exercises* using Jeliot. Most (roughly three fourths) of the findings in the utilization-category were dealing with this matter, as the use was most intensive in Group 2 and could be easily studied in Group 2a and 2b lab groups.

The use of Jeliot was not uniform in the programming exercises. Some students followed the execution through the trace window or by looking at the program outputs. These findings support the argument made by Petre, Blackwell and Green (1994), that designers should not expect that the users would be using their visualization tools in exactly the context they had planned. The tool should be as flexible as possible.

Jeliot was *used independently* by the Group 1 students, although the use was rare. According to the interviews, 4 students out of 11 interviewed had used Jeliot independently. All of them had been using the pre-programmed sorting algorithm examples in Jeliot. It seems that creating programs for Jeliot is hard without introductory tutoring, even though there was a short help text available on the back cover of the learning diary (a leaflet distributed in Group 1) and on the Web.

How Did Jeliot Help Learning?

As stated above, the nature of the study was merely exploring and therefore the learning result was not studied. However, the question of the visualization tool's helpful features was approached from the students' perspective by asking their views of the matter. The material from the observations has been used in this evaluation as well.

Most of the findings indicated that the subjects *had found Jeliot helpful*. Interestingly, the responses indicated that the visualization was useful because of the movement:

> "I think the idea was quite good. Particularly when you see, you can see the idea, that... how the data is being transferred and processed." (Group 1)

> "Those flying boxes (laughing). Well they can first look stupid, but nevertheless they anyway illustrate pretty well what is going on in the program itself." (Group 2)

One pupil underlined that simplicity and the visual features made Jeliot feel like an interesting tool. She added, that this tended to increase her motivation. McWhirter (1996) noticed a similar effect: the use of animation seemed to motivate the students, while Byrne, Catrambone and Stasko (1999) have presented analogous thoughts. These findings support the meta-analysis of Stipek (1996). She presents several studies dealing with computer-assisted instruction (CAI), where even the most simple embellishments (e.g. balloons popping, music, graphics) have been found to have positive effects on motivation.

According to the interviewees, Jeliot is *especially suitable for beginners*. As the tool visualizes manipulation of variables it can support the concept-forming and help in understanding the basic structures. The next level where visualization of variables could be useful comes when the algorithms

are too complex to be easily understood. In Group 1 a lot of effort was make in order to illustrate the structure of the objects.

As the goal of the courses studied was to increase the knowledge of language features and not the complex data processing, especially the Group 2 pupils did not see the visualization relevant for more advanced programmers, who mastered the basic concepts and structures. The only exceptions of the more complex algorithms in Group 1 course were the basic sorting techniques, where the visualization was utilized.

Byrne, Catrambone and Stasko (1999) have also discussed about the subject. According to them the algorithm animation is most valuable for the students with basic knowledge about the domain. However, their experiment differs a lot from the one reported here.

What factors gave trouble to the use of Jeliot?

This study was aimed to record the drawbacks of Jeliot as thoroughly as possible to help the development of Jeliot and similar SV environments. Most of the problems were dealing with user interface. Program bugs were rare and the Jeliot server crashed only once during the term. The client-side software problems were caused by the various incompatible Java-implementations of the browsers. These technical problems did not prevent the use of Jeliot but caused certain annoying instability.

The main *problems in programming* with Jeliot were due to *poor error messages*. In some cases the program did give the line number where the error could be found, but this information was usually invisible below the error message window. The informative error messages and other support are needed especially by the novices who tend to make a lot of syntactic mistakes (Merrill et al. 1995).

The Java version of Jeliot has some minor differences from the standard Java. The need for re-writing programs using standard input and output was too difficult for Group 1 students. For the teacher it was an important reason for rejecting Jeliot in the beginning of the Group 1 course.

The *design of the user interface* needs further development, as can be expected from a program, which is in the prototype stage. The study identified several problems. Generally, the user interface could benefit from re-thinking and perhaps some use of graphical symbols to make it easier to use. The user interface was scattered to four windows, which made the system look complex. The users tended to select all variables to be shown, which now had to be done manually every time after the compilation. In Group 2 the pupils felt uncomfortable because of the English (foreign language to them) user interface and error messages.

Problems *directly related to computer technology* were causing unexpectedly little trouble. Beforehand the unreliable network connections between the high schools and the Jeliot server were expected to pose a significant problem, but this was not the case. At first, Group 2c pupils were trying to use Jeliot via modems, but this was too slow, and therefore too expensive, for them. The most annoying technology-related problem was caused by the instability of some browsers. This incompatibility of different programs or versions was not reliably tested or recorded as the teacher or the observer did not have time for that.

Interestingly, *the visualization itself* seemed to bring a few problems with it. As the lecturer in Group 1 pointed out, the choice of design for the animation window is essential in learning. The data from the observation supports this as the Group 1 teacher and Group 2 students did move the variables on the screen to suit their mental model of the program. Also, the steps of the visualization of the different clauses should be carefully re-thought. Most clauses need several actions to be animated (e.g. for-loops). For an inexperienced learner this can be confusing, as they can not relate the different parts of the clause and the different actions on the screen.

It is also possible that some users did not know how to utilize a visualization tool like Jeliot. Gurka and Citrin (1996) suggest that students should be taught strategies to utilize the visualization tools effectively.

Discussion

This paper presented an outline of the results related to the use of Jeliot in three different teaching environments: at university level, in high school, and in network-based distance education. According to the results visualization is useful, but the prototype of Jeliot needs further development before it can be taken as a serious tool for program visualization in everyday teaching. However, one should remember that Jeliot was originally designed for animating algorithms, not for teaching programming. We have pointed out improvements, which should also be considered by other developers in the field.

Developing Jeliot

The two courses revealed very different views to Jeliot. In Group 1 it was far from a useful tool while in Group 2 it turned out to be a helpful partner for the teacher. The Group 2 course was a short one (10-12 hours) and very little can be said about the learning outcomes. The most important results are certainly in the affective area. After the course the pupils had felt what it is like to teach a computer.

Most of the results confirm existing studies dealing with learning programming and SV in general: the novices would have needed syntax-related support (Merrill et al. 1995), the SV accelerated the learning (Boroni et al., 1996) or was otherwise useful (Elenbogen 1996; Palakal, Myers & Boyd 1998). Considering Jeliot, the user interface and especially the error messages need to be developed. The lack of text input and output features almost completely prevented the use of Jeliot in Group 1. It is evident that these features should be implemented.

The results raised some new ideas which could be useful in utilizing SV applications in learning situations. Visualizations seem to be able to create a fundamental cognitive structure to the learners, and therefore it could be helpful in the orientating phase of the learning.

In tutoring, providing orientating visualizations could help learners to move from human thinking to algorithmic thinking. The problem is how these visualizations could be produced. Should we offer a set of ready-made visualizations of the most fundamental structures, which the tutor or the learner him/herself could use to get support? These visualizations could be utilized before the learner has been able write any code that could be visualized.

The experiences we got from the two groups indicate that the current status of Jeliot forces teachers to rethink their courses. This was not done in Group 1 resulting the lack of use of the tool. Contrary to this, in Group 2 Jeliot was successfully utilized as it was taken into account while planning the course and its assignments.

Is it acceptable that a tool changes the way of teaching and learning? Here, there seem to be two strands for the future development. According the conservative view the tool should be developed to support the existing practices. The motives for such an opinion can be various, for example good feedback of the current way of work, uncertainty in utilizing novel pedagogic ideas or content with existing course material.

Opposite view, a liberal one, would see a tool useful only if it changes existing practices. The most important property of a novel tool is to offer something new for the teaching-learning process and therefore lead to new procedures. In this case it could mean autonomous, interactive study modules in the www, new kind of assignments ("Follow the visualization and implement the algorithm…"), or utilizing small-group discussions promoted by the visualization.

Taking the conservative strand, Jeliot should be developed towards an interactive blackboard. Here, the focus of development should be on improving control tools in viewing visualization: selecting key variables automatically, presenting only certain lines of the program etc. In order to approach this goal we collected information about the illustrations and explaining strategies used in Group 1. These results will be reported elsewhere. According to the liberal viewpoint the priority of development should be in making the user interface easier to use and increasing syntax-related functions (error messages, automatic advisors etc.).

To summarize, the basic question is to decide what to do: either to develop a tool for existing teaching and learning practices, or change teaching and learning practices by developing a new tool.

Can We Automatize Visualization?

According some of the findings the development of the SV tools is not a straightforward issue. McWhirter (1996) suggests that creating visualization should be made as automatic as possible. This has been implemented in Jeliot, but it looks like we now are facing even more difficult problems.

According to the findings the users saw the color and location of the variable symbols relevant. The users designed the visual appearance based on the variables' meaning and importance. So in order to totally automatize SV the tool has to understand the meaning of the variables as well as the relations between them.

Petre, Blackwell and Green (1998) have claimed that there is an inherent contradiction in the SV. It makes programming more concrete, but on the other hand using symbols has a habit of making things even more abstract:

> "It helped me all right. Sometimes the boxes helped me to understand the events. On the other hand, from time to time I got mixed up when I didn't understand what they mean and what is going on." (Group 2)

In this example the complexity of the animation has obviously caused problems in understanding. Creating visualization is not a trivial task. A successful visualization depends on cultural aspects (personal preferences, course habits and wider society-level symbol language).

There seems to be a surprising paradox: If making a good visualization requires familiarity with the variables of the program, how could visualization be helpful in achieving familiarity with strange programs?

References

Boroni, C. M., Torlief, J. E., Goosey, F. W., Ross J. A., and Ross R. J. (1996) Dancing with DynaLab. Proc. of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education, *SIGCSE Bulletin* 28(1), 135-139.

Byrne, M. D., Catrambone, R. and Stasko, J. T. (1999) Evaluating animations as student aids in learning computer algorithms. *Computers & Education* 33(4), 253-278.

Elenbogen, B. S. (1996) Parallel and Distributed Algorithms Laboratory Assignments in Joyce/Linda. Proc. of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education, *SIGCSE Bulletin* 28(1), 14-17.

Gurka, J. S. and Citrin, W. (1996) Testing Effectiveness of Algorithm Animation, *Proc. of 1996 IEEE Symposium on Visual Languages*, September 3-6, 1996. Boulder Colorado. IEEE Computer Society Press, Los Alamitos, 182-189.

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T., and Vanninen, P. (1997) Animation of user algorithms on the Web. *Proc. VL '97, IEEE symposium on visual languages*, IEEE Computer Society Press, Los Alamitos, 360-367.

Järvinen, K., Pienimäki, T., Teräsvirta, T., Kyaruzi, J. J., and Sutinen, E. (1999) Between Tanzania and Finland: learning Java over the Web. *Proc. of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, ACM, New Orleans, 217-221.

Lahtinen, S.-P., Sutinen, E., and Tarhio, J. 1998. Automated animation of algorithms with Eliot. *Journal of visual languages and computing* 9(3), 337-349.

Lincoln, Y. S. and Guba, E. G. (1985) *Naturalistic Inquiry*, Sage, Beverly Hills.

Markkanen, J., Saariluoma, P., Sutinen, E., and Tarhio, J. (1998) Visualization and imagery in teaching programming. In Domingue, J. and Mulholland, P. (eds.), *Proc. 10th annual meeting of the psychology of programming interest group*, Knowledge Media Institute, Open University, Milton Keynes, 70-73.

McWhirter, J. D. (1996) AlgorithmExplorer: A Student-Centered Algorithm Animation System. *Proc. of 1996 IEEE Symposium on Visual Languages*, September 3-6, 1996. Boulder Colorado. IEEE Computer Society Press, Los Alamitos, 174-181.

Meisalo, V., Rautama, E., Sutinen, E., and Tarhio, J. (1997) Teaching algorithms with animation - a case study using Eliot. In Hatakka, O. (ed.), *Proc. LeTTET '96, Learning technology and telematics in education and training*, The Press of University of Joensuu, 79-84.

Meisalo, V., Sutinen, E., and Tarhio, J. (1997) CLAP: teaching data structures in a creative way. *Proc. of the ITiCSE '97, Integrating technology into computer science education*, ACM, Uppsala, 117-119.

Merrill, D. C, Reiser, B. J., Merrill, S. K, and Landes, S. (1995) Tutoring: Guided Learning by Doing. *Cognition and Instruction* 13(3), 315-372.

Palakal, M. J., Myers, F. W., and Boyd, C. L. (1998) An Interactive Learning Environment for Breadth-First Computing Science Curriculum. Proc. of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education, *SIGCSE Bulletin* 30(1), 1-5.

Petre, M., Blackwell, A., and Green, T. (1998) Cognitive Questions in Software Visualization. In Stasko, J., Domingue, J., Brown, M. H., and Blaine, A. P. (eds.), *Software Visualization. Programming as a Multimedia Experience*, The MIT Press, Cambridge Massachusetts, 453-480.

Stasko, J. and Lawrence, A. (1998) Empirically Assessing Algorithm Animations as Learning Aids. In Stasko, J., Domingue, J., Brown, M. H., and Blaine, A. P. (eds.), *Software Visualization. Programming as a Multimedia Experience*, The MIT Press, Cambridge Massachusetts, 419-438.

Stipek, D. J. (1996) Motivation and Instruction. In Berliner, D. C. and Calfee, R. C. (eds.), *Handbook of Educational Psychology*, Simon & Schuster Macmillan, New York, 85-113.

Strauss, A. and Corbin, J. (1991) *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*, the 3rd printing, Sage Publications, Newbury Park.