

# Program Visualization for Programming Education – Case of Jeliot 3

Roman Bednarik, Andrés Moreno, Niko Myller  
Department of Computer Science  
University of Joensuu  
firstname.lastname@cs.joensuu.fi

**Abstract:** Jeliot is a family of program visualization tools. Jeliot 3 is the most recent member of the family. It visualizes the execution of a program by means of animation. Jeliot 3 has been used in different educational settings by diverse users, ranging from students to teachers. Jeliot 3 is published under an open licence and thus it is freely available for anybody to use and develop further. We have conducted several studies and experiments that aimed at improving the system, engaging students with visualization and ultimately at supporting learning of programming. We review the recent research and developments related to Jeliot 3 and discuss possible future steps in its development.

## Introduction

Programming is a complex cognitive domain with several, often implicit and hidden dependencies. The ability to program involves multiple skills, strategies and requires extensive domain and problem knowledge. Because of these properties and characteristics, learning to program and to become a good programmer is difficult. With a hope to make the learning and understanding of programming concepts easier, several algorithm and program visualization tools have been developed in past. Evaluations of the tools in learning contexts, however, have given mixed results both supporting and rejecting the use of software visualizations for learning. Nevertheless, there have been only a few attempts to develop a theoretical framework for the reasons why the program visualization tools are successful or unsuccessful. Hundhausen et al. (2002) have attempted to proceed into this direction. According to Hundhausen's studies one of the key features leading to success is the involvement in which students are committed during the visualization (e.g. watching vs. interacting). Tools that engage students during the visualization support learning more effectively. As a result, the actual form of the visualization does not affect the learning as much as the interaction with the tool.

We and others have conducted both qualitative and quantitative investigations of students' interaction and learning with Jeliot 3. By analysing the current state-of-art, we hope to obtain a benchmark for future improvements in program visualization systems.

In order to engage the user to work with Jeliot 3, it allows automatic animation of the programs and it provides ways of interacting with the visualization. Thus, it allows the learner to explore new concepts and even develop programs with the support of the visualization. Furthermore, Jeliot has been successfully used as a lecturing aid helping the teacher to explain the programming concepts both verbally and graphically. We have

researched the effects of Jeliot 3 on learning and interaction from different perspectives. In the following we review some of the results.

## Jeliot 3

Jeliot 3 (Moreno et al., 2004b) is a program animation tool aimed to support novices in their learning to program (See Figure 1). It is published under the GPL license and is freely available at (<http://cs.joensuu.fi/jeliot/>). Jeliot 3 graphically displays the execution of object-oriented programs written in Java. Most of the Java language concepts are currently supported and animated by Jeliot 3. The model that is visualized in Jeliot 3 represents the actions of a virtual machine during program execution. However, the animation is representing on the Java language level and not the level of bytecode. Thus, the program visualization can be used to teach and learn programming concepts on the introductory level. The visualization (illustrated in the right frame of the Figure 1) shows all the details of the program execution by visualizing expression evaluations, method calls, and object- and array allocations. Another view (not shown in the Figure) shows the call tree of the program. Furthermore, Jeliot provides a possibility to explore the history of the execution in the form of static snapshots taken during the visualization.

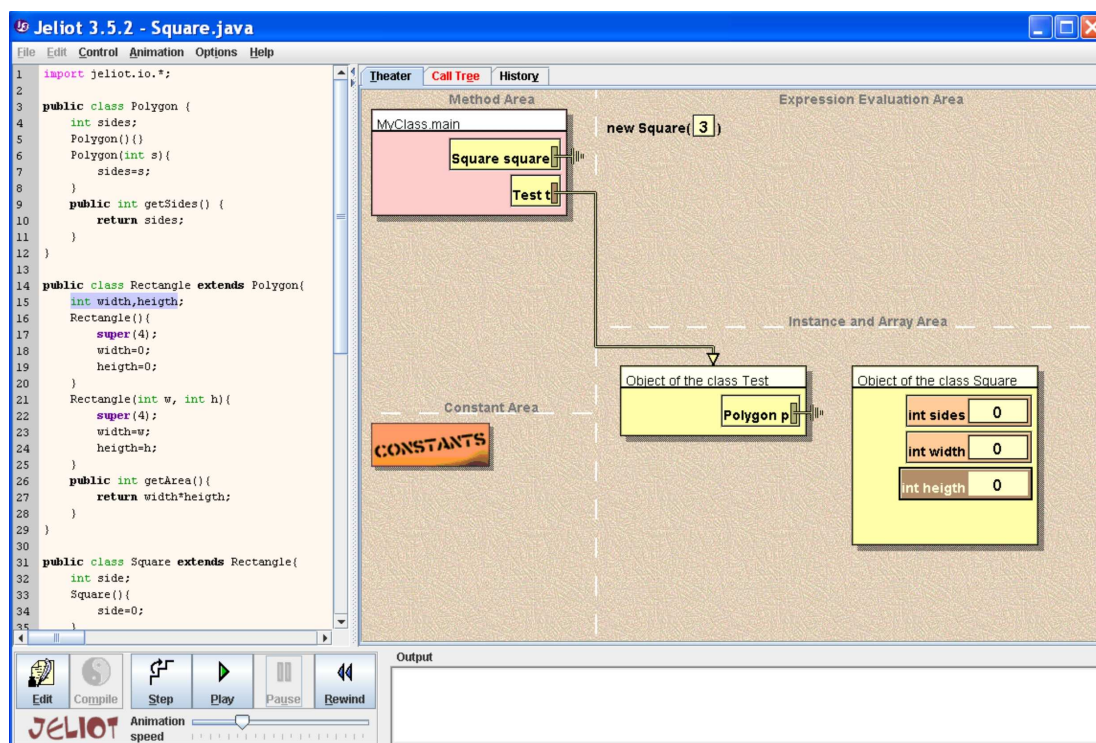
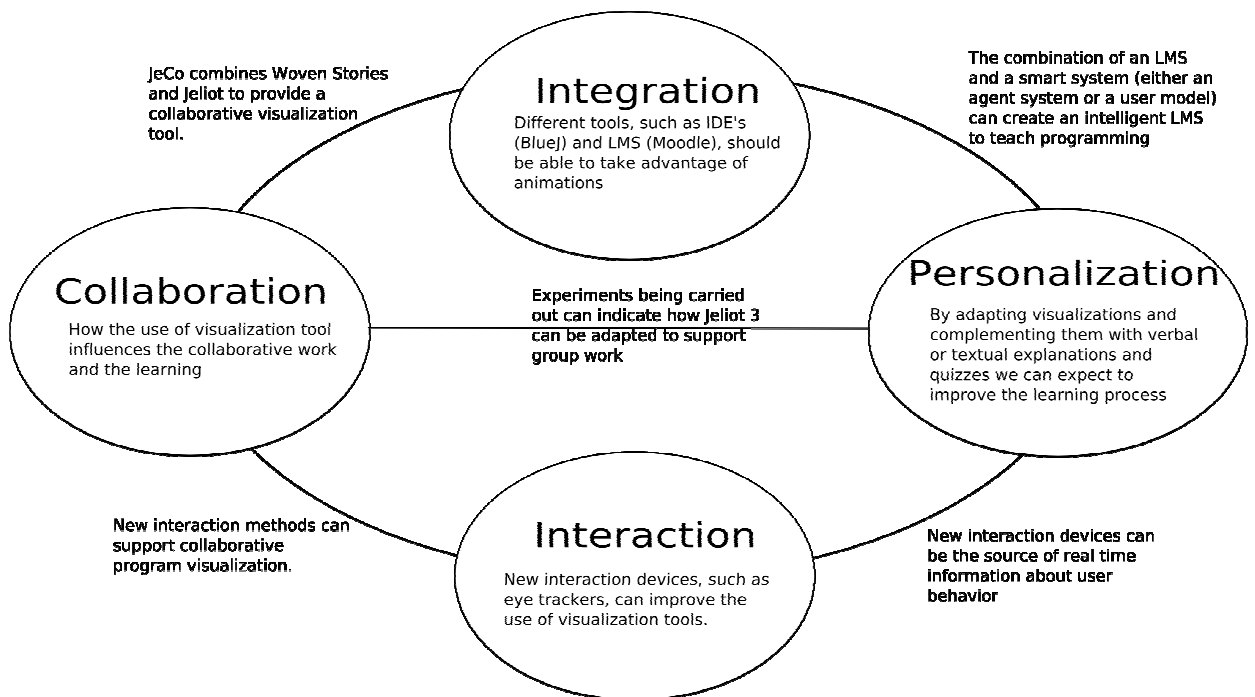


Figure 1. The user Interface of Jeliot 3.

One of the goals that guided the development of Jeliot 3 was to overcome the unnecessary coupling of animation engine and interpreter in the implementation of Jeliot 2000, the previous version of Jeliot family. In Jeliot 3, we separated the animation engine from the interpretation of the source code. Modular design has allowed us to develop new

views of the program execution based on the same interpretation. As a proof of concept, a method call tree view has been developed. In addition to this, we have developed an extension that allows Jeliot 3 to interact with BlueJ, the widely used programming environment for novices. It is possible to start animations in Jeliot directly from the BlueJ's object bench. Another plugin has been developed for Editing Java Easily (EJE), a development environment containing special features that simplify the usage of the tool for novices and allows the use of the tool directly from a web page.

In a classroom study, Ben-Bassat Levy et al. (2003) found that Jeliot 2000 was helpful for mediocre students. Jeliot 2000 helped students to create viable mental models of the program execution and provided them with a vocabulary to describe the execution. Students were also able to answer questions related to unseen situations by drawing a Jeliot-like display and describing the situation with it. Because Jeliot 3 uses the same visualization scheme as Jeliot 2000 and only extends it, we can assume that similar results are achieved when Jeliot 3 is used in a classroom.



**Figure 2.** Current research directions related to Program Visualization and Jeliot 3.

## Research

Our current research focuses on three directions related to program visualization: collaboration, interaction and personalization (Figure 2). We are researching how program visualization can help collaborative programming in education. Eye-tracking is currently used as a data-gathering tool but we devise to use it as a real-time interaction device. Personalizing the contents should be the next step on programming visualization, so students can make most out of the animations.

These three research directions are focused on how to improve the learning of programming by means of Program Visualization. Thus, two main questions guide our research. First, we want to know how students use Program Visualization environments in different learning settings. Secondly, we are interested in what students learn or comprehend in those situations using Program Visualization. Once a clearer picture of students' interaction and learning with Jeliot 3 is obtained, it will be possible to develop Jeliot 3 further in accordance with our findings.

### **Cognitive Aspects of Program Visualization: Eye-tracking Studies**

In many studies of HCI and psychology, eye-movement tracking provided researchers with insights into cognitive processes related to the investigated task. Resting on the hypothesis that the direction of gaze is linked to the visual attention (so called mind-eye hypothesis) and to the top of the information-processing stack, eye-trackers have been employed to unobtrusively collect the point of gaze at any stage of problem-solving or interaction in general. Based on that knowledge, several models of cognitive processing have been proposed (e.g. in reading), new interaction modes proposed (e.g. eye-typing), or individual differences in several domains have been studied (e.g. in visual search). In the studies of computer programming, eye-tracking has not been applied widely. Crosby and Stelovsky (1990) studied individual differences in reading computer programs. It has been found that reading computer programs involves different strategies than reading prose. However, individual strategies varied a lot so it has been difficult to find differences between novices and more experienced programmers. More experienced subjects spent more time on viewing more meaningful areas.

Eye tracking seems to be an effective and non-invasive tool to study cognitive processing during programming. We have conducted several experiments that employed a remote eye-tracker to study visual patterns during comprehension and debugging tasks with Jeliot and other programming environments.

In a comprehension study with Jeliot 3 (Bednarik, Myller, Tukiainen and Sutinen, 2005b) we investigated whether visual behavior of novice and intermediate participants differ during the animation. Almost no effects of experience were found regarding the patterns of visual attention. However, we have found an effect of previous experience on the depth of required processing. In addition, outside the animation, intermediate and novice programmers interacted differently with the tool and exhibited different comprehension strategies. In a debugging study with Jeliot (currently being analyzed), we investigated (among others) whether eye-movement patterns can be informative about the incorrect interpretation of the source code and animation. Recently, we have studied how the role of each representation in the PV window (e.g. code, animation) changes during comprehension (Bednarik and Tukiainen, 2006). Our results suggest that gaze patterns can indicate what features of the source code and animation are important for a current learner. In addition, as the learning and comprehension progress, the roles of the representations change. These changes have been indicated in the relative times spent looking at the different representations at different phases of the comprehension process.

The results from the eye-movement studies provide also initial information about how the gaze direction could be used as a real-time input device for a gaze-aware program visualization tool. In other learning domains, real time gaze has been used for tailoring the content and for communication with a tool during learning (Wang et al., 2006). Similar approach could be taken in future versions of Jeliot: gaze could be used as one of the sources of adaptation (c.f. the next section of this article) and animation could be tailored to each student's needs.

### **Program Visualization Personalization**

As discussed above, Jeliot 2000 animation was useful for mediocre students, as the animation provided the right amount of information for them (Ben-Bassat Levy et al., 2003). However, we believe that both strong students and weak students could benefit from Jeliot 3 animations. Thus we plan to personalize the contents produced by the tool to the different needs and abilities of students.

Users' needs vary depending on how they approach a programming visualization tool. After a qualitative study of students using Jeliot 3 in a four weeks introductory course (Moreno et al., 2006), it became clear that students used Jeliot 3 as a learning aid and as debugger. We observed that some students failed to fully understand some of the animation representations after they have been successfully programming such concepts, or even explicitly taught the meaning of the animation. If they use Jeliot as a learning aid, new material should be added to explain better the process. We consider adding explanations of programming concepts and their representation in Jeliot 3. Another useful source of knowledge could be "stop and think" questions were students predict the result after a certain statement is executed (Naps, 2005). Thanks to Jeliot 3 architecture, this can be done automatically. However, explanations and questions will require modeling the domain and the students' knowledge, so that it can deliver the right question or explanation to each student when required. When students were debugging their own programs, they complained that the animations were too slow and lengthy. They failed to use the breakpoint feature of Jeliot, mostly because they were not aware of it. Thus, a simple adaptation of the user interface would have enabled them to realize the importance of that feature when debugging.

Moreover, to perform both kinds of activities, students would like to visualize only a certain part or concept that they are not sure to understand. To support this, we will investigate the possibility of removing those parts of the animations that are understood, and that the students considered basic. This implies that some modeling of the student progress and domain shall be implemented into Jeliot. The user model would develop as the student progresses using the tool and comprehends the aspects of programming.

A design that supports personalization according to the student's learning style has been proposed (Bednarik et al., 2005a). In that design, Jeliot 3 interacts with a multi-agent educational system. The interaction consists of an agent system providing both the user model and the learning object repository. It will retrieve the learning objects that best fit

student's style. In future, Jeliot 3 will update the user model with the student's usage statistics.

Finally, we are going to carry out a quantitative study with the students in a virtual computer science course. The goal of the study is to investigate whether there is a link between Inductive Reasoning Ability (IRA) and the ability of students to comprehend the animations from Jeliot 3. If such a link is found, it would provide a source of adaptation that depends on the cognitive traits of the student. If we could detect students with low IRA in advance, animations could provide them with more hints on what is happening.

## **Collaboration**

The motivation to study how to support collaborative programming with program visualizations is twofold. Firstly, previous research on algorithm visualization has shown that sharing and discussing algorithm representations have increased the students' learning outcomes (Hübscher-Younger and Narayanan, 2003). The effects of visualization on collaborative programming can be partially extrapolated from this work but research on the collaborative programming and program visualization is also needed. Secondly, pair programming has been found to be a good way to motivate and support students to learn programming (McDowell et al., 2003). However, there are only few tools to support collaborative programming in classrooms and especially in distance education.

The aim of this research is to find an empirically based framework to describe what aspects of program visualization can facilitate collaborative programming and how should a program visualization tool be designed in order to support collaborative activities during programming learning.

We approach these questions from three different perspectives. Firstly, we study the usage of the current program and algorithm visualization tools during collaboration. This tells us how the current visualization tools support collaboration and can point to possible problems that are related to the collaborative use of the tool. One of the preliminary findings is that the amount of interaction between the visualization and learners also affects the amount of interaction between two learners. Secondly, we analyze the current program visualization tools with some analytical framework such as communicative dimensions (Hundhausen, 2005) in order to examine their support for collaboration. Furthermore, we expect to find some areas in these analytical frameworks that current program or algorithm visualization tools do not support at all. Thirdly, we design new tools or modify existing ones based on the empirical and theoretical analyses of the existing program visualization systems in order to facilitate more meaningful collaboration. For example, we have developed a working prototype environment, called JeCo (Jeliot Collaboratively), to support a collaborative programming on web-based courses by combining a collaborative authoring environment Woven Stories and Jeliot (Moreno et al., 2004a).

## Conclusion and Future Directions

Jeliot 3 is an open-source program visualization tool that has been developed now over four years and development still continues in order to support diverse learners in various contexts. In this overview, we have introduced recent results and current research directions into cognitive aspects of programming, adaptivity of program visualization and collaboration issues of programming. Based on the findings, we are constantly improving the program visualization system in order to achieve and support greater engagement, user experience and learning.

In the future, we plan to combine these directions for example by adapting the visualization based on eye-tracking data, or analyzing the collaborative use of visualizations with the help of the gaze data collected from several learners at the same time. In parallel, we are building a learning community around Jeliot 3 and we invite other researchers, teachers and learners to visit our website and get involved in use and development of the tool.

## Acknowledgements

Niko Myller was a visiting researcher at the Massey University from February 10 to April 6 during which a part of this work was done and would like to thank the Department of Information Systems at Massey University for hosting him.

## References

Bednarik, R., Joy, M., Moreno, A., Myller, N., Sun, S., Sutinen, E. (2005a). Multi-Agent Educational System for Program Visualization. In *Proceedings of the International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC'2005)*, 283 - 388.

Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M. (2005b) Effects of Experience on Gaze Behaviour during Program Animation. In *Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05)*, 49–61.

Bednarik, R., and Tukiainen, M. (2006). An Eye-tracking Methodology for Characterizing Program Comprehension Processes. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications (ETRA '06)*. New York: ACM Press, 125–132

Ben-Bassat Levy, R., Ben-Ari, M., and Uronen, P. A. (2003). The Jeliot 2000 Program Animation System. *Computers & Education*, 40 (1), 15–21.

Crosby, M. and Stelovsky, J. (1990). How Do We Read Algorithms? A case study. *IEEE Computer*, 23 (1), 24–35.

Hübscher-Younger, T., and Narayanan, N. H. (2003). Constructive and Collaborative Learning of Algorithms. *SIGCSE Bulletin*, 35 (1), 6–10.

Hundhausen, C. D. (2005). Using End User Visualization Environments to Mediate Conversations: A ‘Communicative Dimensions’ Framework. *Journal of Visual Languages and Computing*, 16 (3), 153–185.

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

McDowell, C., Werner, L., Bullock, H. E., and Fernald, J. (2003). The Impact of Pair Programming on Student Performance, Perception and Persistence. In *Proceedings of the 25th international conference on Software engineering*, IEEE Computer Society, 602–607.

Moreno, A., Myller, N., and Sutinen, E. (2004a). JeCo, a Collaborative Learning Tool for Programming. In *Proceedings of Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04)*, 261–263.

Moreno, A., Myller, N., Sutinen, E., and M. Ben-Ari. (2004b). Visualizing Programs with Jeliot 3. In *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI 2004)*, 373–376.

Moreno, A., Joy, M. S. (2006) Data Collection: Jeliot 3 in a demanding learning Environment. Submitted to *Program Visualization Workshop(PVW2006)*.

Thomas L. Naps, T. L. (2005). JHAVÉ – Addressing the Need to Support Algorithm Visualization with Tools for Active Engagement. *IEEE Computer Graphics and Applications*, 25 (5), 49–55.

Wang, H., Chignell, M., and Ishizuka, M. 2006. Empathic Tutoring Software Agents Using Real-time Eye Tracking. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications (ETRA'06)* New York: ACM Press, 73–78.