https://doi.org/10.17234/INFUTURE.2019.26

Review paper

Software Visualization in Education

Vedran Juričić
Faculty of Humanities and Social Sciences, University of Zagreb, Croatia
vedran.juricic@ffzg.hr

Summary
Software visualization technology provides better understanding and more efficient creation and use of computer software by graphically representing its components, functionality or algorithms. There are various uses and benefits of software visualization, from detecting logical bugs, errors and performance bottlenecks to their use in simulations and e-learning. This paper focuses on algorithm visualization tools, approaches and languages, which show various states, transitions and data structures in more abstract and clearer way than algorithms presented with traditional programming code. It also shows the importance of visualization in learning algorithms, those that are taught at the very beginning of programming courses, as well as advanced cryptographic algorithms. Visualization tools differ in the level of engagement they provide to their users: some of them provide only simplest interactivity, while other provide questions and quizzes, changing input data or constructing custom visualizations. The paper analyses modern and most popular tools and approaches in algorithm visualization and compares their characteristics, advantages and disadvantages in order to show the most important characteristics of today's visualization tools and approaches, and their suitability for different problem areas and scenarios. The paper also analyzes the impact of software visualization on students' knowledge, motivation and efficiency.

Key words: software visualization, algorithm visualization, tools, education, analysis

Introduction

Software visualization encompasses multiple terms like program visualization, algorithm visualization and visual programming. They are commonly used in the literature and although they are similar or related, should not be used as synonyms (Price, Baecker, Small, 1993). A formal definition of program visualization states that it is a mapping from programs to graphical representations (Gruia-Catalin, Cox, 1993). Its goal is to simplify an explanation of a certain program segment, program structure or flow control and requires three participants: the programmer who develops a program, the animator who constructs the mapping and the viewer who observes the representation. On the other hand, visual programming is programming that uses multiple dimensions for transferring and presenting semantics (Burnet, 2001). Typical examples include a time dimension that defines before and after relationships or multidimensional objects like diagrams, icons or sketches that provide additional meaning. A programming language whose syntax consists of visual expressions is called a visual programming language and an environment where a code is written is called a visual programming environment (Ingaiis et al., 2008). Therefore, program visualization and visual programming are two entirely different concepts. In visual programming, the program is written by using graphics, while in program visualization is specified in a regular or textual form and the graphics is only used for clarification and description (Myers, 1986).
Algorithm visualization is a process of visualizing a high-level description of a piece of software (Price, Baecker, Small, 1993). It simplifies the understanding and analysis of programs and its elements by showing various states, transitions and data structures in more abstract and cleaner way than algorithms presented with traditional programming code. This paper analyses modern and most popular or influential tools and approaches in algorithm visualization and compares their characteristics, advantages and disadvantages. Also, it analyses their potential usage and impact in educating students with and without prior programming knowledge.

Visualization in education

The information and communication technologies bring new methods and opportunities for more attractive and productive teaching and learning. They are used to create and study learning materials,

track students' progress, write online quizzes; they increase accessibility of learning materials, provide virtual learning environments and enable a development of student's individual work and qualities by individualization of tasks, where each student can be assigned different problem difficulty or different amount of time needed for its solution (Majhertová, Palásthy, Gunčaga, 2014; Noor, 2013). ICTs have potential to accelerate and deepen students' skills, to motivate and stimulate students, and to make their learning more interesting and engaging.

An important benefit of using ICT in teaching mathematics and informatics is a possibility of visualization, animation and simulation. From student's perspective, visualization provides a new approach for discovering the structure and component dependencies, to solve problems and to discover and analyse results and data, enabling acquisition of new information and knowledge. From teacher's perspective, it provides a new teaching style that complements standard verbal presentation, enabling more dynamics and reducing a time required for explaining a complex problem (Majhertová, Palásthy, Gunčaga, 2014).

A core challenge in teaching programming is helping a student to reason or infer about execution of program code and not to think about the syntax itself. Students should learn how a static textual representation or programming code maps to a dynamic process or program execution (Guo, 2013; Sorva, 2012). Learning programming language is the minor problem in computer courses, where beginners learn about variables, branches and loops. The main problem is how to develop student's thinking and appropriate approaches, and how to use his knowledge of programming language for solving higher level problems. A language, i.e. language syntax and rules, can be learnt in a couple of days, but it can take years to become a good programmer.

Robins et al. (2003) show that the average student does not make much progress in an introductory programming course. Most teachers confess that large number of graduates are beyond standard of programming competence and are not able to program. Guzdial (2011) discovered that only 14% of Yale's students that completed introductory programming course were able to solve the Rainfall problem, a simple problem with reading input numbers and calculating their average. Every study that has used this problem has found similar low performance (Guzdial, 2011). Another study shows that most students of introductory programming courses either fail miserably or pass with extremely high results, with only few that pass with average results (Dehnadi, Bornat, 2006).

The purpose of program visualization is to enhance students' understanding of program behaviour on level higher than understanding programming code. The analysis of three surveys (Naps et al., 2002) based on more than 140000 respondents who were involved in visualization supported programming course showed that 90% find teaching experience more enjoyable and more than 76% of the students showed improved level of participation and motivation. More importantly, for 72% of the students was shown that their learning was improved. Another study was conducted on a control group of 94 students and a test group of 78 student that were using visualization tool. It showed that students achieved better results with visualization, that is they were able to predict program behaviours more accurately.

Although the results of this studies show positive impact of visualization on teaching and learning process, some authors argue that a visualization does not have a significant educational value if it does not engage its users, i.e. students, in an active learning activity (Naps et al., 2002). It can be achieved by writing their own input data sets, predicting future visualization states, programming an algorithm, answering questions or constructing their own visualizations (Hundhausen, Douglas, 2000). Byrne, Catrambone and Stasko (1999) observed that pure visualization and animation does not improve students' knowledge and that there exists no statistically significant difference between animation and PowerPoint presentations, but they confirmed the improvement when student were able to enter their own data into the experiment or program. In other words, in order for a visualization technology to be successfully applied in education, it should not refer to passive images or observations, but must be interactive and enable student to experiment with states and input data.

Visualization tools

Program visualization is studied for many years now and there have been developed numerous tools and platforms that support the learning of programming languages and programming in general. This chapter describes the most popular tools according to the size of their community but also tools that are unique or have special impact on this technology.

UUhistle (Uuhistle.org, 2019) is a program visualization system made for visualizing the execution of small, single-threaded programs in introductory programming courses. It has been designed to support active learning and encourages interaction between a student and visualization. UUhistle can be used in numerous use cases (Sorva and Sirkiä, 2011): animated debugging of programs that students wrote themselves, making presentations as a help when analysing program execution, visual program simulation, interactive mode and program animation, quizzes that can be embedded in example programs etc. System is a subject of many scientific papers and even doctoral thesis because of its amazing interaction and positive effect on students. It is free, written in Java for Python programming language and available for offline usage and as an applet. Unfortunately, it is no longer developed and its latest version was launched in 2013. An example of tool's appearance when visualizing a method invocation is shown in Figure 1.

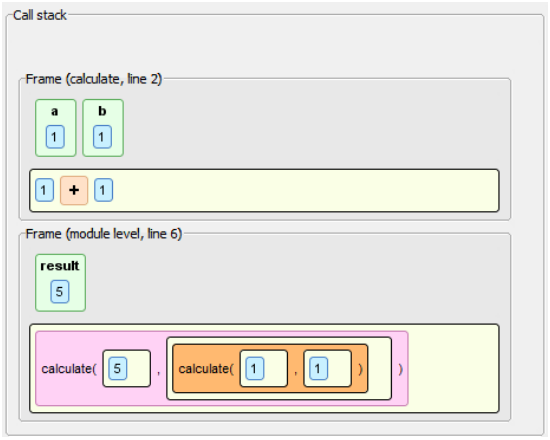

Figure 1. UUhistle showing simple method invocation, source: Uuhistle.org, 2019

Jsvee & Kelmu (Sirkiä, 2018) are two successors of UUhisle system. Jsvee is a JavaScript library that helps teachers to visualize notional machines and create expression-level animations, while Kelmu is a toolkit that enables writing more advanced explanations. They are not developed for a certain language and can be customized and extended. There are numerous online examples for Java and Python programming languages. They are being actively developed and their latest version was launched in August 2019.

Ville (Cs.utu.fi, 2009) is also a program visualization tool that supports multiple programming languages. It comes with syntax and example editor, but teachers can also write, define and add their own languages, which enables its maximum extensibility. Students can edit code, view call stack and variables, set breakpoints and have complete control over visualization and animation. It also integrates quizzes so that students can test their knowledge of observed problem. It is free, written in Java and available as desktop version and as an applet.

Python tutor (Pythontutor.com, 2013) is an advanced program visualization tool that supports Python, Java, C, C++, JavaScript and Ruby programming languages. It has a large user community, including students and instructors, with over five million people in 180 countries. It provides interactive environment where students can experiment with program execution, including live programming mode, where students can analyse recursions and complex programming problems. It stands out with live help module, supported by many volunteers around the world, that allows anyone to join user's session and help him debug program by writing code or explaining a problem using integrated chat. Python tutor is free and accessible through a web application. Figure 2 shows an example of recursion visualization.

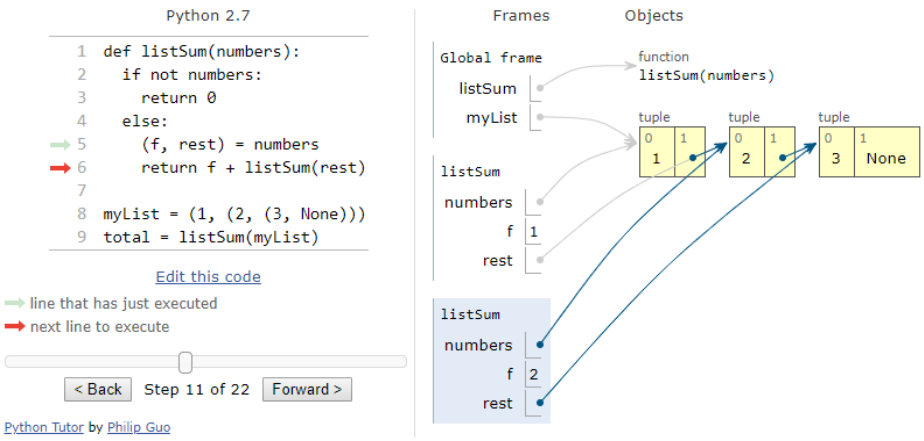

Figure 2. Python tutor visualizing a recursion, source: Pythontutor.com, 2013

BlueJ (Kouznetsova, 2007) is a tool that differs from above described ones because it is not designed as an aid for learning basics, but for object-oriented programming, which is more difficult for students than procedural programming. It is integrated development environment for Java programming language, that comes with textbook, teacher support and detailed documentation. The tool is open-source, under GNU General Public licence and available on Windows, Linux and MacOS operating systems. It offers an overview of object activities and their dependencies and provides an interaction with objects and methods, including their invocation with custom parameters. An example of object and its methods is shown in Figure 3.

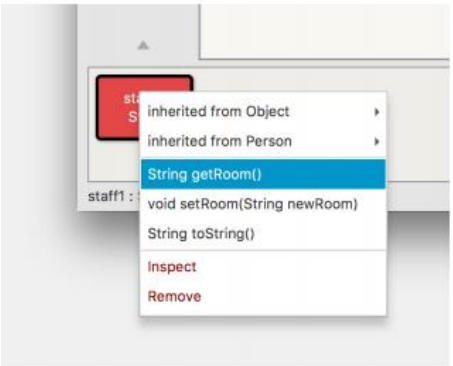

Figure 3. BlueJ displaying possible method invocations

Tools that were described in this paper are only a small subset of available ones. There exists over 50 program visualization tools that naturally differ in their features and possible usages. Most of them share on important characteristic, interactivity, which is in accordance with results of previously conducted studies that showed its positive effect on students' satisfaction and knowledge. They have the option to perform step-by-step analysis of target program execution and experiment with input data and current memory state, like variable values on stack and heap. Most of them are free, written by developer community and with partially available source code. They commonly support two programming languages, Python and Java. Currently they are the most represented languages in education, Python in introductory programming courses and Java in object-oriented programming.
Tools differ in their access and run methods. Some of them are available only as a standalone application for Windows, MacOS or Linux, and some of them through web browser, which gives them features like possibility to save examples in the cloud, to share them with community and to request help and tutoring. The most obvious difference is a result of their purpose; some of them are created as an aid with basic programming, some of them for algorithm visualization and some of them show dependencies on a higher level, like objects and their interconnections.

Conclusion

This paper describes software visualization taxonomy and shows advantages and suggestions for using program visualization tools in education. It is shown that using visualization technology is not sufficient in order to improve students' knowledge; it only improves their satisfaction and interest for programming. The real progress occurs when visualization is broadened with interactivity, including experimenting with input data, variables and program flow. Paper describes some of the available visualization tools that are often used in teaching programming, in order to show their common features and characteristics.

References

Burnett, M. (2001). Visual Programming. Wiley Encyclopedia of Electrical and Electronics Engineering

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

Cs.utu.fi. (2009). VILLE - the visual learning tool. Available at: https://ville.cs.utu.fi/old/?p=2 (10.9.2019)

Dehnadi, S. and Bornat, R. (2006). The camel has two humps (working title). Middlesex University, UK, 1-21

Gruia-Catalin, Cox, K. (1993). A Taxonomy of Program Visualization Systems. // Computer 26, 12, 11-24

Guo, P. J. (2013). Online python tutor: embeddable web-based program visualization for cs education. // Proceeding of the 44th ACM technical symposium on Computer science education, 579-584

Guzdial, M. (2011). From Science to Engineering – Exploring the Dual Nature of Computing Education Research. Communications of the ACM 54, 2, 37-39

Hundhausen, C. D., Douglas, S. (2000). Using Visualizations to Learn Algorithms: Should Students Construct Their Own, or View an Expert's? IEEE Symposium on Visual Languages, Los Alamitos, California, 21-28

Ingaiis, D., Wallace, S., Chow, Y.-Y., Ludolph, F., Doyle, K. (2008). Fabrik A Visual Programming Environment. // ACM SIGPLAN Notices 23, 11

Kouznetsova, S. (2007). Using BlueJ and Blackjack to teach object-oriented design concepts in CS1. // Journal of Computing Sciences in Colleges 22, 4, 49-55

Majhertová, J., Palásthy, H., Gunčaga, J. (2014). Educational Software and Visualization in Teaching. Future Learning

Myers, B. A. (1986). Visual programming, programming by example, and program visualization: a taxonomy. // ACM SIGCHI Bulletin 17, 4, 59-66

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C. and Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. // ACM Sigcse Bulletin 35, 2, 131-152

Noor, S. U. (2013). An Effective use of ICT for Education and Learning by Drawing on Worldwide Knowledge, Research, and Experience: ICT as a Change Agent for Education. // Scholarly Journal of Education 2, 4, 38-45

Online Python Tutor (2013). Embeddable Web-Based Program Visualization for CS Education. Philip J. Guo. ACM Technical Symposium on Computer Science Education (SIGCSE)

Price, B. A., Baecker, R. M., Small, I. S. (1993). A Principled Taxonomy of Software Visualization. // Journal of Visual Languages & Computing 4, 3, 211-266

Pythontutor.com. (2013). Python Tutor - Visualize Python, Java, C, C++, JavaScript, TypeScript, and Ruby code execution. Available at: http://pythontutor.com/ (11.9.2019)

Robins, A., Rountree, J. and Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. // Computer Science Education 13, 2, 137-172

Sirkiä, T. (2018). Jsvee & Kelmu: Creating and tailoring program animations for computing education. // Journal of Software: Evolution and Process, 30, 2, e1924

Sorva, J. (2012). Visual program simulation in introductory programming education. Aalto University

Sorva, J. and Sirkiä, T. (2011). Context-sensitive guidance in the UUhistle program visualization system. // Proceedings of the 6th Program Visualization Workshop (PVW'11), 77-85

Uuhistle.org. (2019). UUhistle.org. Available at: http://www.uuhistle.org (10.9.2019)