Editorial: Program and Algorithm Visualization in Education

GUIDO RÖSSLING
Technische Universität Darmstadt
and
J. ÁNGEL VELÁZQUEZ-ITURBIDE
Universidad Rey Juan Carlos

This special issue presents extended versions of six papers presented at the 5th Program Visualization Workshop (PVW'08). The articles deal with many of the issues relevant to program and algorithm visualization in education. This foreword introduces these issues to better understand the challenges addressed by every article, and their relevance, as well as the articles featured. These issues are evaluation of program and algorithm visualization, integration of visualization and narratives into hypertextbooks, embedding of interactive quizzes into visualizations, and several classes of program visualization.

Categories and Subject Descriptors: K.3.1 [**Computers and Education**]: Computer Uses in Education; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*

General Terms: Algorithms, Experimentation, Human Factors

Additional Key Words and Phrases: Program visualization, algorithm visualization, computer science education

ACM Reference Format:
Rößling, G. and Ángel Velázquez-Iturbide, J. 2009. Program and algorithm visualization in education. ACM Trans. Comput. Educ. 9, 2, Article 8 (June 2009), 6 pages.
DOI = 10.1145.1538234.1538235. http://doi.acm.org/10.1145.1538234.1538235.

1. INTRODUCTION

The Program Visualization Workshop series has been organized in Europe every second (even) year since 2000. The aim of this workshop series is to bring together researchers who design and construct visualizations or animations, as well as visualization or animation systems, for computer science, mainly for

Author's address: G. Rößling, Technische Universität Darmstadt, CS Department Hochschulstr. 10, D-64289 Darmstadt, Germany; email: guido@tk.informatik.tu-darmstadt.de; J. Á. Velázquez-Iturbide, Universidad Rey Juan Carlo, Departamento de Lenguajes y Sistemas Informáticos 1, c/ Tulipán s/n, 28933 Móstoles, Madrid, Spain; email: angel.velazquez@urjc.es.

programs, data structures, and algorithms. Above all, the workshop attracts educators who create, use, or evaluate visualizations and animations in their teaching.

In this issue, we present extended versions of six papers presented at the 5th Program Visualization Workshop (PVW'08) in Madrid, Spain. The articles featured in this issue of TOCE are good representatives of the kinds of issues that are currently subject to active research in the field of software visualization.

Software visualization is a broad term that includes many different classes of visualization. Price et al. [1998] made a distinction between program and algorithm visualization that is broadly accepted. Program visualization refers to graphical representations to enhance human understanding of the actual implementation of programs. Algorithm visualization refers to graphical representations of the higher descriptions of programs (algorithms) that are later implemented as programs.

We find proposals to visualize software since the 1960s, but it is in the early 1980s when algorithm animation as an educational resource comes to the mainstream of scientific and technical research. Intuitively, most educators agree about the great potential of software visualization for computer science education. However, adoption of software visualization is lower than developers of program or algorithm visualization (PAV) systems would expect.

There are a number of reasons for this lack of acceptance by users, especially by educators. A survey conducted among educators by the ITiCSE 2002 Working Group on "Improving the Educational Impact of Algorithm Visualization" [Naps et al. 2003] determined the following most frequently cited factors that made educators reluctant or unable to use software visualizations:

—Lack of time for different tasks (e.g., to learn a new tool or to develop a visualization).
—Technical issues of software visualization tools (e.g., lack of effective development tools or lack of reliable software).
—Integration into the courses or the classroom (e.g., time to adapt visualizations to a course or visualizations may hide important details or concepts).
—Other factors (including lack of evidence of educational effectiveness).

The ultimate goal of educational PAV systems is to aid students to learn more effectively. Therefore, the success of a particular system can only be proved by evaluation. Many systems are available, but there have been fewer evaluations of these systems than one might have expected. Some articles in the literature have surveyed effectiveness of software visualization. Probably the most important study was performed by Hundhausen et al. [2002], concluding that the way students use visualizations is more important than the visualizations themselves.

Partly based on the conclusions of Hundhausen et al. [2002], the ITiCSE 2002 Working Group cited above proposed a taxonomy of learner engagement [Naps et al. 2003]. The taxonomy differentiates six levels, which from lower to higher are: *no viewing, viewing, responding, changing, constructing*, and

presenting. It also assumes that the higher the level of engagement with a visualization, the higher the learning.

The articles contained in this issue deal with many of these issues. We provide a brief overview of the articles in the next section.

2. AN OVERVIEW OF THE ARTICLES

The first article, "A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems," reviews and analyzes the literature on educational uses of software visualization. Many articles that describe program or algorithm visualization systems focus on technical issues. Being an important issue, we have explained above how important other issues are to promote software visualization in education. The authors Urquiza-Fuentes and Velázquez-Iturbide focus on evaluations that yielded positive results, considering 18 PAV systems that were subject to a total of 33 evaluations. To facilitate the analysis, PAV systems are first classified according to two general criteria: abstraction level—either program or algorithm visualization—and implementation approach.

This article surveys two kinds of evaluations. The first class of evaluations is applicable to any interactive software system, namely usability. The authors find that usability evaluations are shallow in almost all the cases, and are hardly integrated into the system development process. The second class of evaluations is applicable to any educational system, namely educational success. The article also identifies the engagement levels achieved in each evaluation and additional system features present in each evaluation. Specific features present in positive evaluations are identified and it is suggested to make more use of them. As future work, an analysis of the evaluations which did not yield educational success is necessary to extrapolate these results to a conclusive meta-study on the educational success of PAV systems.

Three articles in this issue deal with pedagogical features of PAV systems. The article "Seamless Integration of Hypertext and Algorithm Animations" by Karavirta deals with the issue of integrating animations and narratives, or more generally with hypertext. Given the universal role of the Web, the author proposes to integrate animations into dynamic Web pages. He first states requirements for an algorithm animation system based on experience accumulated in past years and made explicit in the literature. These requirements are used by Karavirta to design an algorithm animation viewer. The viewer implements most of these requirements, resulting in a versatile tool for learning. In particular, it supports customizing and changing animations, responding to quizzes, interplay of hypertext and animations, and other features.

This article illustrates how educational requirements are used to guide the design of an animation tool and also as a checklist for the interested user. The article is also interesting for technical developers. The viewer was developed with simple but powerful technology, namely HTML and JavaScript, and is capable of viewing animations in the XAAL algorithm animation language. The interplay of the different parts of the system are explained in several parts

of the article. The article also gives JavaScript code details to better illustrate the system functionality.

The article "A Visualization-Based Computer Science Hypertextbook Prototype" by Rößling and Vellaramkalayil is related, but addresses the integration with hypertext within the context of the course management system Moodle. As with the previous article, the authors first state the requirements for their integration, here based on an ITiCSE Working Group report. As a consequence of adopting Moodle, the hypertextbook is modeled as a Moodle activity. The working prototype illustrates many of the features of visualization-based hypertextbooks.

The article is of interest to the same audience as the previous one. In particular, it is very interesting to instructors on showing the features a future hypertextbook may exhibit. This interest is current given the great acceptance of Moodle. It is also interesting to the technical developer, as it describes the different elements involved and the process necessary to make it work. It shows details of dialogs and configuration files, so that even the non-initiated can understand them.

Bruce-Lockhart, Pierluigi, and Norvell are the authors of "Adding Test Generation to the Teaching Machine." Their article presents a contribution at the engagement level of responding. The authors present a new extension of the Teaching Machine environment, called the Quiz Generator. The extension allows generating multiple-choice quizzes of five types. The quizzes make use of the visualization capabilities of the Teaching Machine to aid the student in understanding the questions.

The article is interesting to two audiences. First, it contains a categorization of quizzes that will be of interest to educators. The authors propose a definition for a "testable algorithm" that provides a model where different types of quizzes can be defined. As a consequence, it is easy to recognize the constituent elements of each type of quiz and the functionality that has to be added to a system (here, the Teaching Machine) to implement it. Second, the article outlines the implementation of the Quiz Generator that will be of interest to developers of visualization systems. The implementation is described in general terms as flow of information or functionalities needed, although details are also given, for example, scripting commands.

Finally, two articles deal with program visualization. The article "Robust Generation of Dynamic Data Structure Visualizations with Multiple Interaction Approaches" by Cross et al. is an interesting effort to produce program animations close to the algorithmic level. They introduce visualizations of data structures delivered by the jGRASP IDE, called "dynamic viewers." The user can interact with the visualizations in three ways: the debugger, the workbench, and a text-based interaction tab. The three approaches are complementary and can be used together at the user's convenience. An interesting contribution of their dynamic viewers is the fact that, although they produce program visualizations automatically, the IDE uses a "structure identifier" in an attempt to identify the traditional visualization that best fits the actual data structure, for example, a stack or a queue. As a consequence, the abstraction level of the program visualization is close to the algorithmic level.

The authors give a brief description of the implementation of the three interaction approaches. However, they do not emphasize the technical details but rather the adequacy of their development. For this end, they provide two evaluations. The first one measured the accuracy with which the structure identifier correctly identified the adequate visualizations. Notice that such accuracy is critical to their approach to more abstract visualizations. As students are often trained with examples extracted from textbooks, the structure identifier was subject to code extracted from 20 textbooks. The second evaluation was composed of four controlled experiments with students in lab sessions. They were aimed at measuring the effect of using the dynamic viewers on students' performance. Two of the experiments focused on whether students wrote new code for a given problem more accurately and in less time. The other two experiments focused on whether students would be able to detect and correct logical errors (as well as introducing fewer new logical errors) more accurately and in less time. Their results are very positive in both evaluations.

The article "Compiler Optimization Pass Visualization: The Procedural Abstraction Case" shows a less common usage of program visualization. The abstraction level at which its visualizations are aimed is much lower, as they display optimization in machine language code. In particular, a program map visualization highlights the pieces of code where instructions have been abstracted as a subroutine call. This visualization consists in a sequence of pixels, each one representing either an instruction or a memory byte. Using a coloring convention, the user may visually detect in the visualization the fragments of code abstracted.

The article provides an example of a simple but non-intuitive visualization, that must be explicitly taught to its users (here, students). The authors' proposal was evaluated by means of an opinion questionnaire given at the end of a class session. The students' answers show that they found the visualizations useful and motivating.

The papers by Cross et al. and Rößling and Vellaramkalayil shared the PVW 2008 Best Paper Award.

ACKNOWLEDGMENT

We want to thank our reviewers for PVW 2008 and for this issue for their excellent work:

—Mordechai Ben-Ari (Weizmann Institute of Science, Israel)
—Pierluigi Crescenzi (University of Florence, Italy)
—Camil Demetrescu (University of Rome "La Sapienza", Italy)
—Ari Korhonen (Helsinki University of Technology, Finland)
—Lauri Malmi (Helsinki University of Technology, Finland)
—Thomas L. Naps (University of Wisconsin Oshkosh, USA)
—Rockford J. Ross (Montana State University, USA)
—Jaime Urquiza-Fuentes (Universidad Rey Juan Carlos, Spain)

REFERENCES

HUNDHAUSEN, C., DOUGLAS, S., AND STASKO, J. 2002. A meta-study of algorithm visualization effectiveness. *J. Vis. Lang. Comput. 13*, 3, 259–290.

NAPS, T., RÖSSLING, G., ALMSTRUM, V., DANN, W., FLEISCHER, R., HUNDHAUSEN, C., KORHONEN, A., MALMI, L., MCNALLY, M., RODGER, S., AND VELÁZQUEZ-ITURBIDE, J. 2003. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull. 35*, 2, 131–152.

PRICE, B., BAECKER, R., AND SMALL, I. 1998. An introduction to software visualization. In *Software Visualization*, J. Stasko, J. Domingue, M. Brown, and B. Price, Eds. Cambridge, MA: MIT Press. 3–27.

Received April 2009; accepted April 2009