Visualizing the Execution of Programming Worked-out Examples with Portugol

Isabel C. Moura

***Abstract*—Learning to program is hard. But it can be facilitated for novice undergraduates. The programming task requires them to master, for instance, the solutions for standard problems and the meaning of running programs. Completing programming worked-out examples directs these students' attention to learning the essential of relations between problem-solving moves. Hence, the learning of programming solutions was supported by this program-completion approach in both the 2010 and 2011 editions of a computer science introductory module at the University of Minho. The learning of running worked-out examples can be further assisted by a program visualization tool. This pilot study reports the changes verified after introducing the Portugol tool for students to automatically visualize the execution of programming worked-out examples in the 2011 edition of that same module; and compares those changes to the 2010 edition. The positive significant effect on students' achievements, which made them rise, is then showed and discussed.**

***Index Terms*—computer science education, novice programmers, program visualization, worked-out examples**

I. Introduction

STUDENTS who choose to graduate in Polymers Engineering Integrated Master (PEIM) – a five-year degree program at the University of Minho (UM), have to pass the two-module Programming and Numerical Methods (PNM9703) course. Programming is a Computer Science (CS) introductory module of this second year course of PEIM studies. Learning to program entails acquiring and developing complex knowledge and skills [1], [2], [6], [8], [9], [17], [21]. Some examples follow on what the programming task requires from these novice undergraduates. They are supposed to master [4], [14], [15]:

--The solutions for standard problems.

--The meaning of running programs.

--The general idea of programs.

--A programming language.

--The skills of planning, developing, testing, and debugging programs.

Active learning techniques may help students accomplish these points, as these techniques keep them highly involved in the learning process [8], [9], [12], [13]. But, providing minimal guidance to novice undergraduates (e.g., by making them generate the solutions for programming problems) puts a heavy load on their working memory. (The human working memory has limited capacity for dealing with new information.) The latter prevents some of these students from learning CS fundamentals. Worked-out examples can help mitigate this problem. Studying and further completing worked-out examples (e.g., solutions for standard programming problems) directs novice students' attention to learning the essential of relations between problem-solving moves, reducing the cognitive load on their working memory [7], [15], [18], [19].

The programming module of the PNM9703 course covers two thirds of the semester. So, the material taught consisted of programming basic constructs (e.g., variables, assignment statements, selections, loops, and arrays). In 2011 and 2010, in-class active instructional activities (see, e.g., [5], [11], [16]) were used to introduce these CS fundamentals. That is, during each lab session the instructor presented a standard programming problem and led the students to build the corresponding algorithmic solution; then, they were supposed to code, test, and debug it. In addition, students were handed over a set of short, textbook-type algorithmic segments of 1 to 30 lines long (tops) (i.e., worked-out examples that solved standard programming problems). These examples started by being complete and flawless. As the weeks progressed, flaws and missing lines were increasingly added for students to complete and/or correct. Throughout the programming module students were supposed to study, complete, and/or correct each worked-out example and to code, test, and debug it. They were also requested to find out the general idea for each example (i.e., the programming problem). Assessment consisted of two individual tests. These tests (mainly of multiple-choice questions) aimed at evaluating students' recognition of syntactic errors and understanding of the structure and function of simple algorithmic and code sequences [21]. This program-completion approach seemed to facilitate the learning of solutions for standard programming problems (e.g., [7]–[9], [12], [13], [15], [18], [19], [21]).

Research in CS education (for a detailed review, see [15]) argues that novice undergraduates also find it hard to mentally simulate the execution of a program; and thus, understanding the meaning of running programs. To address this problem, the literature on program animation for training purposes (e.g., [2]–[4], [14], [17]) suggests instructors to introduce these students to a simple

Manuscript received March 07, 2013; revised March 30, 2013. This work was supported in part by FEDER Funds under program "Programa Operacional Fatores de Competitividade – COMPETE" and Portuguese Funds, by the "FCT – Fundação para a Ciência e Tecnologia" under Grant FCOMP-01-0124-FEDER-022674.

I. C. Moura is with the Centro Algoritmi, Universidade do Minho, 4800-058 Guimarães, Portugal (phone: 351-253-510266; fax: 351-253-510300; e-mail: icm@dsi.uminho.pt).

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

description of the machine they are learning to operate (e.g., the procedural notional machine); and use a program visualization tool to support this description. It also suggests instructors to give students basic programming tasks so they can interact with the tool. According to these same authors, such use of these tools can help novice undergraduates build a clear mental model of the execution of programs, by showing them the hidden mechanics of the notional machine. The more students deepen their understanding about the execution of programs, the more likely they are to succeed in learning CS fundamentals.

Portugol Integrated Development Environment (IDE) 2.3 was then incorporated in the learning environment of the 2011 edition of the PNM9703 programming module; and students were further required to use it. This program visualization tool allowed them to automatically animate procedural algorithmic solutions written in a Portuguese pseudo-code like language. So, having a stable version of the Portugol IDE, students were supposed to: (i) automatically format a given algorithmic solution (or worked-out example) (i.e., color and indent the pseudo-code); (ii) automatically check the latter for syntactic errors; (iii) correct them; (iv) run/test the syntactically correct algorithmic solution step-by-step while monitoring the corresponding change of the internal state of the variables; (v) edit the solution as needed; (vi) repeat steps (i) to (v) until they got a complete and flawless solution for a standard programming problem.

This pilot study reports on the impact that the implementation of the PNM9703 programming module (at the UM in the fall semester of 2011), entailing the use of a tool for students to automatically visualize the execution of programming worked-out examples, had on their academic achievements. Empirical results regarding the effectiveness and pedagogical benefits of visualization tools are mixed (for an overview, see [2], p. 376-377). However, since the Portugol IDE 2.3 was integrated into an environment that tends to facilitate learning [3], [7], [9], [12], [13], [18], [19], a positive effect was expected. Method and results are then presented. A discussion follows on this study's results and potential ways of improving such an implementation.

II. Context and Research Design

Programming is hard to learn although being generally accepted (at the UM) as a useful skill for engineering students (e.g., [1], [2], [6], [9], [14], [15]). Plus, the PEIM studies only had a programming module (part of PNM9703 – a second year mandatory course, with no prerequisites, offered in the fall semester) exclusively dedicated to the development of CS fundamentals. Given the short term (two thirds of the semester) for the module, the instructor's main concern was to avoid over expose students to content. This gave them the opportunity to interact with the content in a meaningful way and avoided blocking the learning [20]. Reference [1] also suggests that the procedural paradigm is more appropriate than the object-oriented one to teach programming fundamentals to novice undergraduates. So, the CS material aimed at the PNM9703 programming module was reduced to its basic constructs (e.g., variables, assignment statements, selections, loops, and arrays) and taught in accordance with the procedural paradigm.

Both editions (i.e., the 2011 and 2010) of the PNM9703 programming module under scrutiny consisted of a weekly 130-minute lab session. In-class active instructional activities were used (see, e.g., [1], [5], [8], [11], [16]). That is, students started each session by being lectured (for approximately 5-15 minutes) on the algorithmic constructs intended for the solution of a standard programming problem (refer to Appendix A). Then, they were asked to put together (individually or in groups of two) an algorithmic solution in a couple of minutes (i.e., students practiced the knowledge lectured). Right after, one of the students' solutions was written, discussed, and improved on the board. This was done by having the instructor: (i) showing the students how to manually trace the execution of an algorithm; (ii) asking 'what-if' questions; (iii) letting the students work on their answers and presenting them before class. (CS fundamentals previously taught were revisited, as needed.) In the remainder of each lab session, undergraduates were supposed to study, complete, and/or correct textbook-type algorithmic solutions (or worked-out examples) of 1 to 30 lines long (tops). (Flaws and missing lines were increasingly added to these solutions throughout the module.) In addition, students were guided through the programming language text book so they could code, test, and debug (individually or in groups of two) the algorithmic solutions. Students were also asked to summarize the general idea for the solutions (i.e., to find out the programming problem being solved). At home, students were supposed to finish the worked-out examples started in class. Visual Basic was the adopted programming language. The MS Excel 2007 VBA programming environment was chosen. (This environment made it easy for students to automate the handling of datasheets they work with during the PEIM program.)

Yet, many novice undergraduates are unable to write a piece of code by the end of a whole semester practicing programming [9]. So, the two individual tests consisted mainly of multiple-choice questions; and were aimed at evaluating students' recognition of syntactic errors and understanding of the structure and function of simple algorithmic and code sequences [21]. The first test (i.e., a multiple-choice test) covered material on variables, assignment statements, selections, and 'while' loops. In addition, the second test (which also covered 'for' and 'do-until' loops and arrays) required students to: (i) fill-in the gaps for a simple algorithmic and/or code segment; and (ii) write a simple piece of code equivalent to a given one. Overall grades for the programming module of the PNM9703 course were derived 40% from the first test and 60% from the second test. Students took the first and second tests after attending four and eight lab sessions, respectively.

The 2010 edition of the PNM9703 programming module aimed at facilitating the learning of solutions for standard programming problems, by using the reported program-completion approach [7], [9], [15], [19], [21]. But, students were also supposed to manually trace the execution of worked-out examples (just like the instructor did during lab sessions, as she used the call stack to describe the execution of procedural algorithmic solutions). This required them to

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

mentally simulate the execution of examples and imagine the dynamic behavior and side-effects of running examples. However, many novice undergraduates found these activities extremely difficult. Consequently, in 2010, students had a hard time understanding the meaning of running programs (whether written in a pseudo-code or programming language) [2]–[4], [14], [15], [17].

To help students overcome this difficulty, in 2011 they were required to automatically animate worked-out examples (or algorithmic solutions), by using Portugol IDE 2.3. (The tool interface is presented in Fig. 1.) That is, firstly, students used the tool editor (i.e., the large upper window right below the pull-down menu in Fig.1) to write an example and automatically format it. (The "automatic format" option in the *Editar* pull-down menu (see Fig. 1) automatically colors and indents the pseudo-code, which makes it easy to read.) Second, students were supposed to use the "verify" option in the *Algoritmo* pull-down menu (see Fig. 1) to automatically check the example for syntactic errors. (This option highlights pseudo-code lines that have errors in the editor screen and provides feedback on each error – one at a time). Third, students had to correct the syntactic errors reported by the tool using the "verify" option as needed (i.e., until they got an error-free solution). Finally, students were required to run/test the example (free from syntactic errors) using the "*Executa e Monitora*" option in the *Algoritmo* pull-down menu. (This option opens a new window with two vertical frames, i.e., the "*Executa e Monitora*" window in the centre of the screen in Fig. 1). By repeatedly pushing the right button on top of the left frame (for continuing with the execution of the next statement), students were able to execute an example step-by-step at their own pace and visualize (on the right frame) the effect of each statement on the internal state of the variables. This step-wise animation allowed students to form and explore their own hypothesis (as they inserted input data, e.g.) and draw conclusions for the examples [2], [4], [14]. After a few lab sessions, some students got bored with this way of running examples. They were then told to use the left button on top of the left frame (i.e., the "50%" button of the "*Executa e Monitora*" window in Fig. 1). The "50%" button and the cursor (located right below this button on the left frame of the "*Executa e Monitora*" window in Fig. 1) allowed students to establish the slow-motion speed at which Portugol IDE 2.3 showed them (on the right frame) the automatic step-wise execution of the example and the corresponding update of the internal state of the variables. The instructor gave students feedback on their solutions and corresponding visualizations, as needed. In general, making students interact with a program visualization tool (like Portugol IDE) increases their engagement with it; and thus, enhances their understanding of the execution of procedural solutions and their mental models for the procedural notional machine [2], [4], [14]. Students were introduced to Portugol IDE 2.3 in the beginning of the module and taught how to use it.

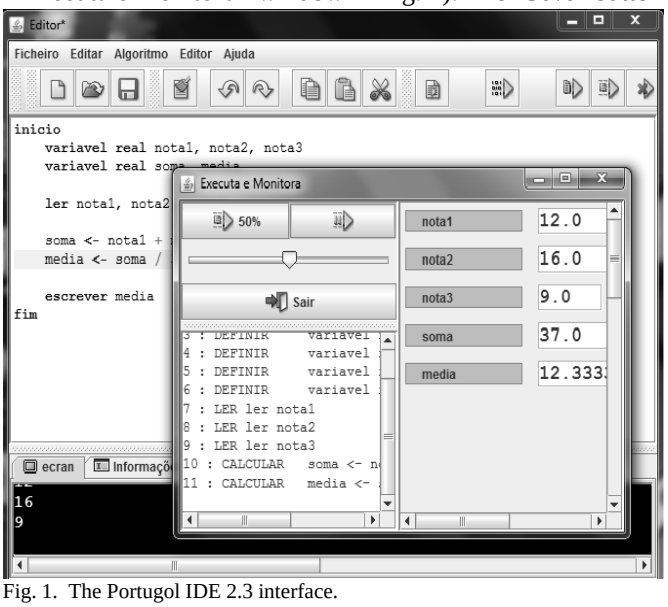

Fig. 1. The Portugol IDE 2.3 interface.

TABLE I
STUDENTS' MEANS AND STANDARD DEVIATIONS FOR THE FIRST AND SECOND TEST GRADES BY SEMESTER

	First test	
	Fall 2011 (N = 31)	Fall 2010 (N = 22)
Mean (SD)	11.7 (3.28)	10.2 (3.86)
Minimum	6.2	3.2
Maximum	18.5	16.8
	Second test	
	Fall 2011 (N = 31)	Fall 2010 (N = 22)
Mean (SD)	13.7 (3.13)	11.1 (2.93)
Minimum	5.4	6.8
Maximum	19.4	15.6

Portugol IDE 2.3 is a freeware environment for training procedural paradigm fundamentals [10]. It is a standalone application that can be downloaded from the Portugol website (http://www.dei.estt.ipt.pt/portugol) and easily installed on a personal computer. The tool interface is simpler than Jeliot's (refer to [2], p. 378) but, fairly similar to it. Overall, Portugol IDE 2.3 is a simple, intuitive, and stable IDE that enables novice undergraduates (on their own) to create, edit, develop, test, and automatically animate algorithmic solutions. These solutions must be written in a Portuguese (i.e., students' native language) pseudo-code like language (refer to Appendix B), which is quite similar to the one taught in the 2010 edition of the PNM9703 programming module. Portugol IDE 2.3 pseudo-code language is built around a small number of constructs and kept simple in its syntax and semantics. This program visualization tool has been used by Portuguese and Brazilian higher education institutions.

III. Research Questions and Methodology

Literature on program animation for training purposes [2]–[4], [14], [15], [17] suggests that if novices are provided with: (i) a simple description on how the notional machine works; (ii) an easy to use program visualization tool that illustrates the mechanics of it; (iii) the possibility of completing basic programming tasks using the tool, they will be able to build a clear mental model of the execution of solutions for standard programming problems. The more students deepen their understanding about the meaning of running programs, the more likely they are to succeed in

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

learning CS fundamentals. This hypothesis raised the following research questions:

1) Are there differences in students' achievements (on average) for the first individual test between the 2011 and 2010 editions of the PNM9703 programming module?
2) Are there differences in students' achievements (on average) for the second individual test between the 2011 and 2010 editions of the PNM9703 programming module?

In the current study quantitative methodologies were used in the analysis and interpretation of data.

IV. Data Collection and Results

Data spanning two semesters were collected. Students who had previously been exposed to a similar CS content were excluded from both the 2011 and 2010 samples. (An improvement in these students' grades was expected.) This means that a total of 63 students (i.e., 35 in the fall of 2011 and 28 in the fall of 2010) attended the programming module of the PNM9703 course for the first time. Students who dropped the module in both semesters (even though a few of them have taken the first test) were also excluded. Consequently, data from a total population of 53 students (i.e., 31 from the 2011 edition and 22 from the 2010 edition) were examined. These novice undergraduates were from the second year of PEIM studies.

The averages of students' achievements (on a 0-20 scale) for the first and second individual tests by semester are summarized in Table I.

Overall, Table I results seem to show an improvement for the class that attended the 2011 programming module of the PNM9703 course. That is, the highest grade averages for the first and second tests were received by those who studied in the 2011 edition of the module. Examining the first research question, the *Mann-Whitney* test result on the equality of mean ranks ($Z = -1.67$, $p\text{-value} < 0.10$) suggests that the 2011 students' first test grade average is numerically and marginally statistically different from the 2010 students' first test grade average. Therefore, the 2011 undergraduates might have started taking advantage of Portugol IDE 2.3 (after four weeks of using it for running increasingly difficult worked-out examples) to better understand the information presented in lab sessions. Concerning the students' achievements for the second test (i.e., the second research question), the *Mann-Whitney* test result on the equality of mean ranks ($Z = -2.78$, $p\text{-value} < 0.01$) indicates that the 2011 students' second test grade average is numerically and statistically different from the 2010 students' second test grade average. This significant improvement for the 2011 students' performance suggests that the programming module approach that used the Portugol IDE 2.3 facilitated the development of an appropriate mental model of the execution of procedural solutions for standard programming problems. Therefore, an improvement in the learning of CS fundamentals might have occurred in the 2011 edition. However, it took about eight weeks (of practicing with the referred program visualization tool) for these undergraduates to accomplish the latter result.

V. Discussion, Conclusions, and Recommendations for Future Research

This pilot study reports on the use of a tool for students to automatically visualize the execution of programming worked-out examples in an undergraduate CS introductory module of the PNM9703 course at the UM in 2011. This module implementation comprised:

--In-class active instructional and learning activities for solving standard programming problems and tracing the execution of corresponding algorithmic solutions.

--Using a program visualization tool (i.e., Portugol IDE 2.3) for novice undergraduates to automatically animate worked-out examples (i.e., short, textbook-type algorithmic solutions for standard programming problems that were handed over complete and flawless, in the beginning, and increasingly incomplete and/or flawed as the module progressed) that they were supposed to study, complete, and/or correct.

--Coding, testing, and debugging the referred algorithmic solutions.

--Two individual test assignments consisting mainly of multiple-choice questions.

Both individual test grade averages for the 2011 programming module of the PNM9703 course indicate that students responded favorably to the use of Portugol IDE 2.3. That is, given Table I results, this implementation of the module (in the 2011 edition) provided a better learning environment for novice undergraduates to build a clear mental model of the execution of procedural solutions for standard programming problems, compared to the 2010 implementation. Therefore, students were more likely to succeed in learning CS fundamentals [2]–[4], [14], [15], [17]. Note that students took some time (about eight weeks) to learn the program visualization tool (i.e., Portugol IDE 2.3) and to fully benefit from its use. That is, the substantial (and statistically significant) improvement in the 2011 test grade averages (compared to the 2010 ones) was reported for the second individual test only. This result resembles the findings of [3]'s. These authors' approach to teaching and learning CS fundamentals differed from the one reported here. But, the program visualization tool they used (i.e., Jeliot 2000) was fairly similar to Portugol IDE 2.3.

To conclude, the current study suggests that, in CS introductory modules, instructors can provide an enhanced learning environment aimed at improving novice undergraduates' academic achievements. This environment entails (at least) facilitating the learning of: (i) solutions for standard programming problems, by using a program-completion approach; and (ii) the meaning of running programs, by using a program visualization tool. However, the successful integration of the tool into such an environment depends on the following course of action [2]–[4], [14], [17]. Instructors shall, first, pick a stable, easy to learn and use tool. Second, introduce students to the tool in the beginning of the module. Third, make sure that students use the tool throughout the module by, for instance, giving them basic programming tasks (e.g., worked-out examples for them to study, correct and/or complete); and remind

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

them (as needed) that they will be tested on the understanding of the structure and function of pseudo-code sequences structurally identical to the ones trained in class. Finally, instructors shall explicitly teach students how to use the tool and to interpret its automatic visualizations (in the beginning and later on in the module, as needed) by, for instance, making students run worked-out examples step-by-step at their own pace and giving them feedback on the examples and the corresponding step-wise animations.

It can be argued that the undergraduates who participated in this study have the same background, since they are all second year students of the PEIM program. But, the author could not control for the membership of the students to each semester. So, future implementations entailing automatic visualizations of the execution of worked-out examples in CS introductory modules shall provide further insight into students' background and characteristics. That is, qualitative data shall be gathered on: (i) students' overall demographics (e.g., sex, age, and experience and efficacy with computers) and programming experience; (ii) students' perceptions and attitudes towards CS and the program visualization tool (e.g., satisfaction with the tool); and (iii) how students use the tool. With this data, future studies may aid the case that the effects reported here on students' performance are from the tool and not artifacts of the composition of the different classes.

Future studies shall also use a larger population that will help to further validate the significance of the results obtained.

Appendix A: an Example of a Standard Programming Problem (translated into English)

Write a program that computes the average of three given grades for a student.

Appendix B: an Example of an Algorithmic Solution

Version 1.1	
inicio *variavel real nota1, nota2, nota3* *variavel real soma, media*	Declaration of variables
ler nota1, nota2, nota3	Data input
soma <- nota1 + nota2 + nota3 *media <- soma / 3*	Computation
escrever media *fim*	Data output

Acknowledgment

The author thanks several anonymous reviewers for their helpful comments and suggestions.

References

[1] M. Barak, J. Harward, G. Kocur, and S. Lerman, "Transforming an introductory programming course: from lectures to active learning via wireless laptops," *Journal of Science Education and Technology*, vol. 16, no. 4, pp. 325–336, 2007.

[2] M. Ben-Ari, R. Bednarik, R. Levy, G. Ebel, A. Moreno, N. Myller, and E. Sutinen, "A decade of research and development on program animation: the Jeliot experience," *Journal of Visual Languages and Computing*, vol. 22, no. 5, pp. 375–384, 2011.

[3] R. Ben-Bassat Levy, M. Ben-Ari, and P. Uronen, "The Jeliot 2000 program animation system," *Computers & Education*, vol. 40, no. 1, pp. 1–15, 2003.

[4] J. Bennedsen and C. Schulte, "BlueJ visual debugger for learning the execution of object-oriented programs?," *ACM Transactions on Computing Education*, vol. 10, no. 2, pp. 8:1-8:22, 2010.

[5] R. Felder and R. Brent (2009). Active learning: an introduction. *ASQ Higher Education Brief* [Online]. *2(4)*. Available: http://www.asq.org/edu/2009/08/best-practices/active-learning-an-introduction.%20felder.pdf

[6] A. Forte and M. Guzdial, "Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses," *IEEE Transactions on Education*, vol. 48, no. 2, pp. 248-253, 2005.

[7] P. Kirschner, J. Sweller, and R. Clark, "Why minimal guidance during instruction does not work: an analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching," *Educational Psychologist*, vol. 41, no. 2, pp. 75-86, 2006.

[8] M. Linn and M. Clancy, "The case for case studies of programming problems," *Communication of the ACM*, vol. 35, no. 3, pp. 121-132, 1992.

[9] R. Lister, "After the gold rush: toward sustainable scholarship in computing," in *Proc. 10th Conference on Australasian Computing Education*, Wollongong, 2008, pp. 3–17.

[10] A. Manso, C. Marques, and P. Dias, "Portugol IDE v3.x: a new environment to teach and learn computer programming," in *Proc. IEEE EDUCON Education Engineering*, Madrid, 2010, pp. 1007–1010.

[11] J. McConnell, "Active learning and its use in computer science," in *Proc. 1st Conference on Integrating Technology into Computer Science Education*, Barcelona, 1996, pp. 52–54.

[12] M. Prince and R. Felder, "The Many Faces of Inductive Teaching and Learning," *Journal of College Science Teaching*, vol. 36, no. 5, pp. 14–20, 2007.

[13] M. Prince and R. Felder, "Inductive teaching and learning methods: definitions, comparisons, and research bases," *Journal of Engr. Education*, vol. 95, no. 2, pp. 123–138, 2006.

[14] H. Ramadhan, F. Deek, and K. Shihab, "Incorporating software visualization in the design of intelligent diagnosis systems for user programming," *Artificial Intelligence Review*, vol. 16, no. 1, pp. 61–84, 2001.

[15] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: a review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.

[16] K. Smith, S. Sheppard, D. Johnson, and R. Johnson, "Pedagogies of engagement: classroom-based practices," *Journal of. Engr. Education*, vol. 94, no. 1, pp. 87–101, 2005.

[17] P. Smith and G. Webb, "The efficacy of a low-level program visualization tool for teaching programming concepts to novice C programmers," *Journal of Educational Computing Research*, vol. 22, no. 2, pp. 187–215, 2000.

[18] J. Sweller and G. Cooper, "The use of worked examples as a substitute for problem solving in learning algebra," *Cognition and Instruction*, vol. 2, no. 1, pp. 59-89, 1985.

[19] J. van Merriënboer, P. Kirschner, and L. Kester, "Taking the load off a learner's mind: instructional design for complex learning," *Educational Psychologist*, vol. 38, no. 1, pp. 5–13, 2003.

[20] M. Weimer, *Learner-centered teaching. Five key changes to practice.* San Francisco, CA: Jossey-Bass, 2002.

[21] S. Wiedenbeck, "Novice/expert differences in programming skills," *International. Journal of Man-Machine Studies*, vol. 23, no. 4, pp. 383–390, 1985.