Computer Science Instruction Assisted by a Visualization Tool

Isabel C. Moura

***Abstract*—One of the several difficulties novice undergraduates experience in applying programming fundamentals is mastering the meaning of running programs. Because of these difficulties, students lack involvement for Computer Science (CS) introductory courses; and the latter are associated with high drop-out rates. Integrating a program visualization tool into an environment that tends to facilitate learning helps novice undergraduates to build a clear mental model for understanding the behavior of running programs. This may improve the involvement of students for those courses. Using two different editions of the same CS introductory module, this pilot study portrays the changes from an unassisted to a visualization tool assisted program-completion approach. The results in terms of success, failure, and drop-out are given and the impact of introducing this tool on student involvement in learning is analyzed. The author discusses implications of the assisted implementation for the classroom and pays attention to some of its drawbacks.**

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

I. Introduction

At the University of Minho (UM), students who chose to graduate in Polymers Engineering Integrated Master (PEIM), which is a five-year degree program, must 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. Because of the difficulties novice undergraduates face mainly in applying programming fundamentals [9], some of them seem to get less involved in this module over time. The fairly high drop-out rate associated with the 2010 edition of it may explain part of the problem.

Constructivist based theories have demonstrated that effectiveness of learning is largely dependent on the ability to promote the immersion of the student in authentic situations. In this view, learning to solve problems is a process of individual and collaborative exploration addressed to real context of implementation [10], [21]. Thus, the relevance of CS introductory courses is often revealed as students attempt to solve reality bond problems. Active learning in CS instruction acknowledges these views through, for instance, (i) program-generation approaches (that, e.g., emphasize the design and coding of new programs, to solve problems, with minimal guidance being provided from the lecturer) and/or (ii) the introduction of concepts, methods, and skills on a need-to-know basis in the context of challenge questions. These techniques keep students highly involved in the learning process and enable them to take responsibility for learning [6], [12], [17], [18], [21], [26]. However, cognitive load theory argues that program-generation approaches provided with minimal guidance during instruction put a heavy load on novices' working memory. This prevents some of them from learning to apply CS fundamentals, as the human working memory has limited capacity for dealing with new information. Lecturers can thus facilitate learning by making novices study and further complete solutions (or worked examples) to standard programming problems. Such program-completion approaches direct students' attention to learning the essential of relations between problem-solving moves, reducing the cognitive load on their working memory [8], [20], [23], [24].

In the two editions of the programming module of PNM9703 course (i.e., fall semesters of 2011 and 2010), in-class active instructional activities were used to introduce programming basic constructs, such as, variables, selections, and loops (e.g., with students being presented, in the beginning of each lab session, to a standard programming problem and led to build the corresponding algorithmic solution that they were supposed to code, test, and debug later on during the session [15]. Examples of in-class active instructional activities can be found in, e.g., [5], [13], [21].). These activities help lecturers to involve students in the learning process and shift part of the responsibility for learning to the students. In addition, a program-completion approach was used to facilitate the learning of solutions to standard programming problems (e.g., [8], [11], [12], [17], [18], [20], [23], [24], [27]). This program-completion approach emphasized the completion of worked examples – or short, textbook-type algorithmic segments of 1 to 30 lines long (tops) – that started by being complete and flawless, with flaws and missing lines being increasingly added for students to complete and/or correct as weeks progressed [15]. Despite being regarded as facilitating learning, this environment seemed to fall short for some of the 2010 novices who ended-up dropping the module.

Research in CS education (for a detailed review, see [20]) suggests that applying programming fundamentals requires more than just mastering solutions to standard problems from novice undergraduates. It also requires them to master the meaning of running programs, which entails the ability to mentally simulate the execution of programs [4], [19]. To

Manuscript received March 04, 2014; revised March 24, 2014. This work was supported in part by FCT – Fundação para a Ciência e Tecnologia within the Project Scope: PEst-OE/EEI/UI0319/2014.

I. C. Moura is with the ALGORITMI Research Centre, Universidade do Minho, 4804-533 Guimarães, Portugal (phone: 351-253-510266; fax: 351-253-510300; e-mail: icm@dsi.uminho.pt).

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

promote the learning of this skill, experts in program animation for training purposes (e.g., [2]–[4], [19], [22]) suggest lecturers to introduce novices to a simple description of the machine they are learning to operate (e.g., the procedural notional machine) and use a program visualization tool to assist this description. Furthermore, they suggest lecturers to give students basic programming tasks to make them interact with the tool, and thus, enhance their engagement with it. Such use of these tools helps novices to build a clear mental model for understanding the execution of programs, by showing them the hidden mechanics of the notional machine. The more students deepen their understanding (and mental models) about the meaning of running programs (and, e.g., the procedural notional machine), getting involved in learning activities, the more likely they are to succeed. A stronger involvement among students may lead to higher achievements in CS introductory courses (and modules). This hypothesis can be related to the fundamental principles set forth in constructivist learning theory [6], [16]–[18], [21], [26]. However, regarding effectiveness and pedagogical benefits of visualization tools, empirical studies show mixed results (for an overview, see [2], p. 376-377). On the other hand, literature in educational research indicates that the above referred positive effect is expected if program visualization tools are integrated into an environment that tends to facilitate learning [3], [8], [12], [17], [18], [23], [24].

In the 2011 edition of the programming module of PNM9703 course a stable version of a program visualization tool (i.e., Portugol Integrated Development Environment (IDE) 2.3) was integrated into the learning environment. The tool was required for novices to automatically animate procedural algorithmic solutions (or worked examples written in a Portuguese pseudo-code like language). That is, using Portugol IDE 2.3 novices were supposed to (i) automatically format a given algorithmic solution (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 variables, (v) edit the solution as needed, and (vi) repeat steps (i) to (v) until they got a complete and flawless solution to a standard programming problem [15].

This pilot study reports on the impact that the implementation entailing the use of Portugol IDE 2.3 (for students to automatically visualize the execution of worked algorithmic solutions) in the 2011 programming module of PNM9703 course (at the UM) had on its drop-out rate and students' academic achievements. Method and results are then presented. A discussion follows on this study's results and potential ways of improving the referred implementation in a CS introductory module.

II. Context and Research Design

PNM9703 was a second year mandatory course, with no prerequisites, offered in the fall semester. Its programming module, which covered two thirds of the semester, was the first of PEIM studies exclusively dedicated to computational literacy (e.g., involving the ability to create computational artifacts). Given the module's short term, during its 2011 and 2010 editions only programming basic constructs (e.g., variables, assignment statements, selections, loops, and arrays) were taught in accordance with the procedural paradigm. (Reference [1] suggests that the latter is more appropriate than the object-oriented one to teach programming fundamentals to novice undergraduates.) Over exposing students to content was thus avoided, as it may impede learners' meaningful interaction with the content and block the learning [26]. Although the learning environment of both editions (that is described below) tends to facilitate learning, particularly of solutions to standard programming problems [8], [12], [20], [24], [27], it failed to involve some of the 2010 students.

During weekly 130-minute lab sessions (of both the 2011 and 2010 programming module of PNM9703 course) in-class active instructional activities were used to introduce CS fundamentals (for examples see, e.g., [1], [5], [11], [13], [21]). In each session, to start with, a standard programming problem (refer to the Appendix) was presented to students and they were lectured (for approximately 5-15 minutes) on algorithmic constructs meant for the solution. 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 with the lecturer (i) showing students how to manually trace the execution of an algorithm, (ii) asking 'what-if' questions, and (iii) letting students work on their answers and presenting them before class. (CS fundamentals previously taught were revisited, as needed.) In the remainder of lab sessions, undergraduates were supposed to study, complete, and/or correct textbook-type algorithmic solutions (or worked 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 (on Visual Basic under MS Excel 2007 VBA environment, which made it easy for them to automate the handling of datasheets they work with throughout PEIM studies) 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 behind each solution (i.e., to find out the programming problem being solved). At home, students were supposed to finish the worked examples started in class. Assessment consisted of two individual tests and aimed at evaluating students' recognition of syntactic errors and understanding of the structure and function of simple algorithmic and code sequences [27]. The first test (consisting only of multiple-choice questions) covered material on variables, assignment statements, selections, and 'while' loops. Besides answering multiple-choice questions, in the second test (that also covered 'for' and 'do-until' loops and arrays) students had to (i) fill-in the blanks for a simple algorithmic and/or code segment and (ii) write a simple piece of code equivalent to a given one. (According to [12], as many novice undergraduates are unable to write a piece of code by the end of a whole semester practicing programming, multiple-choice questions are good for testing their knowledge of basic constructs.) Overall grades of the programming

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

module of PNM9703 course were derived 40% from the first test and 60% from the second test [15].

During the 2010 edition of the programming module of PNM9703 course students were also supposed to manually trace the execution of worked examples (just like the lecturer did during lab sessions, as she used the call stack to describe the execution of procedural algorithmic solutions). This required them to mentally simulate the execution of examples and imagine the dynamic behavior and side-effects of running examples. As many novice undergraduates found this tracing activity particularly challenging, they skipped it, and thus, had a hard time understanding the meaning of running worked algorithmic solutions (written in Portuguese – or students' native language – pseudo-code like language) [2]–[4], [19], [20], [22]. Conversely, hands-on computing and receiving immediate feedback (e.g., from an IDE program) in and out of class, is perceived by students to have a positive effect on their understanding of programming activities [1]. Therefore, in the 2011 edition of that same module a stable version of Portugol IDE 2.3 (i.e., a program visualization tool) was integrated into the learning environment for students to automatically animate worked algorithmic solutions.

Portugol IDE 2.3 is a freeware environment for training programming fundamentals compliant with the procedural paradigm [14]. 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 presented in Fig. 1. It is fairly similar to but simpler than Jeliot's (refer to [2], p. 378). 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 (or worked examples). These solutions must be written in a Portuguese pseudo-code like language (e.g., refer to the solution printed in the large upper window right below the pull-down menu in Fig.1), which is quite similar to the one taught in the 2010 programming module of PNM9703 course [15]. Portugol IDE 2.3 pseudo-code language is built around a small number of constructs and kept simple in its syntax and semantics [14]. This program

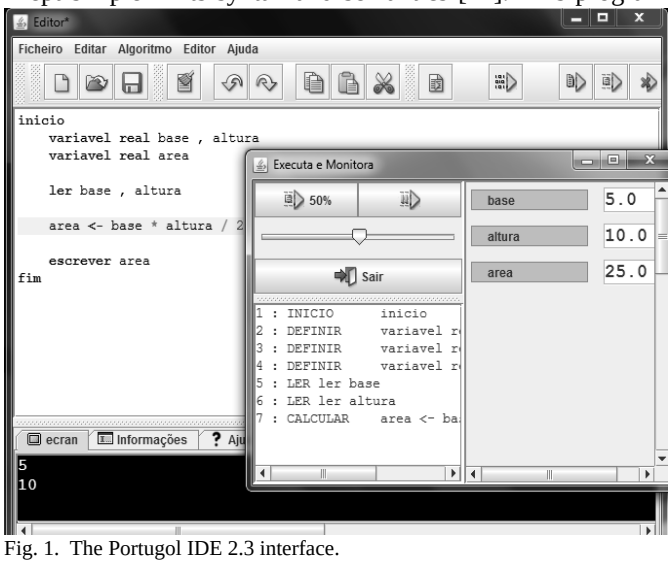

Fig. 1. The Portugol IDE 2.3 interface.

visualization tool has been used by Portuguese and Brazilian higher education institutions.

Novices were introduced to Portugol IDE 2.3 in the beginning of the 2011 programming module of PNM9703 course and taught how to use it. As making students interact with a program visualization tool increases their engagement with it [2], [4], [19], PEIM novices were given basic programming tasks to exploit the tool. First, they used the tool editor (i.e., the large upper window right below the pull-down menu in Fig.1) to write and automatically format examples. (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, novices were advised to use the "verify" option (as needed, e.g., until they got a syntactic error-free solution) in the *Algoritmo* pull-down menu (see Fig. 1) to automatically check examples for syntactic errors. (This option highlights, one at a time, pseudo-code lines that have syntactic errors in the editor screen and provides feedback on each error). Third, students had to correct/remove syntactic errors reported by the tool from examples. Finally, they were required to run/test syntactic error free examples 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 of the "*Executa e Monitora*" window in Fig. 1) the effect of each statement on the internal state of 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], [19]. After a few lab sessions, some of the students got bored with this way of running examples. These students were then taught to use the left button and cursor located on top of the left frame (i.e., the "50%" button and cursor right below this button on the left frame of the "*Executa e Monitora*" window in Fig. 1). Displacing the cursor students established the slow-motion speed at which Portugol IDE 2.3 showed them, after they have pushed the "50%" button, the automatic step-wise execution of an example and the corresponding update of the internal state of variables (on the right frame of the "*Executa e Monitora*" window in Fig. 1). The lecturer gave students feedback on their solutions and corresponding visualizations, as needed [15].

III. Research Questions and Methodology

Literature in educational research argues that integrating an easy to use program visualization tool into an environment that tends to facilitate learning (one that combines, e.g., active learning and program-completion approaches) and that engages students with the tool (e.g., giving them basic programming tasks to make students use it), helps novices to build a clear mental model for understanding the execution of programs. The more students deepen their understanding about the meaning of running programs, getting involved in learning activities, the more

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

likely they are to succeed [2]–[4], [8], [12], [19], [20], [22]–[24]. This stronger involvement among students may lead to higher achievements in CS introductory modules [6], [17], [18], [21]. This hypothesis raised the following research questions:

1) Are there differences in course approval, failure, and drop-out rates between the programming module of PNM9703 course assisted by a visualization tool taught in the fall of 2011 and the unassisted one taught in the fall of 2010?
2) Are there differences in approved students' final achievements (on average) between the programming module of PNM9703 course assisted by a visualization tool taught in the fall of 2011 and the unassisted one taught in the fall of 2010?

In this pilot study quantitative methodologies were used in the analysis and interpretation of data. These methodologies consisted of examining the potential differences for both editions of the programming module of PNM9703 course (i.e., visualization tool assisted and unassisted implementations) in terms of totals of approvals, failures, and drop-outs and approved students' final achievements.

IV. Data Collection and Results

Both the 2011 and 2010 classes of PNM9703 course were composed without the author's intervention (i.e., in the UM's usual manner). Then, data spanning these two semesters were collected on undergraduates registered in the course. Students who had previously been exposed to a similar CS content were excluded from the sample, as an improvement in these students' grades was expected. Thus, data from a total of 63 novices (i.e., 35 from the fall 2011 and 28 from the fall 2010, who attended the programming module of PNM9703 course for the first time) were examined. Given students' academic index, this population had the same background since its undergraduates were all from the second year of PEIM studies.

In the fall semester of 2011, out of 35, 32 students were approved (i.e., 91% of approvals) and three students

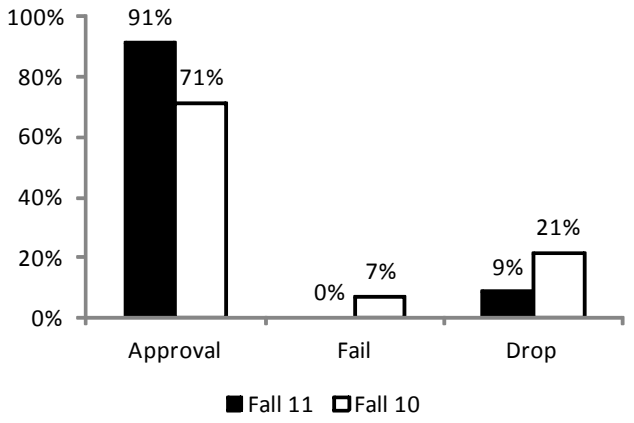

Fig. 2. Approval, failure, and drop-out rates of both programming modules of PNM9703 course offered in 2011 and in 2010.

dropped the programming module of PNM9703 course (i.e., 9% of withdrawals). Surprisingly, there were no failures. According to Fig. 2, both drop-out and approval rates of the programming module suggest that undergraduates might have responded favorably to the implementation assisted by Portugol IDE 2.3 (the program visualization tool). For students involved in this implementation: the drop-out rate was below half (9%) the one of fall 2010; the approval rate was 20 percentage points higher than (91%) the one of fall 2010; and the failure rate reached the lowest value possible (0%).

Regarding the first research question, the test result for the proportion of withdrawals (using small-sample statistics) indicates that the drop-out rate of the programming module of PNM9703 course offered in the fall of 2011 (N = 35) is numerically and marginally statistically different (with *p-value* < *0.10*) from the drop-out rate of fall 2010 (N = 28). This result resembles the one on drop-out attained by [7], who have also used active learning techniques but, a different program visualization tool (namely Turtlet). Similar tests were performed on both proportions of approvals and failures of the same 2011 module. Results suggest that the approval rate of the programming module offered in 2011 is numerically and statistically different (with *p-value* < *0.01*) from the one of fall 2010. This result outperformed the one attained by [7]. But, the failure rate of that same module is not statistically different from the one of fall 2010.

Concerning students' achievements, the final grade average for the approved ones was equal to 13 (SD = 2.05; Maximum = 18; Minimum = 10; N = 32), on a 0-20 scale, in the fall of 2011. In 2010, the final grade average for the approved students of the programming module of PNM9703 course equaled the 12 mark (SD = 2.06; Maximum = 16; Minimum = 10; N = 20). Examining the second research question, no statistically significant differences between semesters were found in the distribution of the approved students' final programming grades.

V. Discussion, Conclusion, Recommendations, and Future Research

This pilot study reports on the use of a tool for students to automatically visualize the execution of worked algorithmic solutions in an undergraduate CS introductory module of PNM9703 course at the UM in 2011. This module implementation comprised (i) in-class active instructional and learning activities for solving standard programming problems and tracing the execution of corresponding algorithmic solutions, (ii) using a program visualization tool (i.e., Portugol IDE 2.3) for novice undergraduates to automatically animate worked examples (i.e., short, textbook-type algorithmic solutions to 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, (iii) coding, testing, and debugging the referred worked algorithmic solutions, and (iv) two individual test assignments consisting mainly of multiple-choice questions [15].

The results of the 2011 implementation for the programming module of PNM9703 course indicate that students responded favorably to the integration of Portugol

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

IDE 2.3 into the learning environment. That is, given Fig. 2 results, making novice undergraduates interact with Portugol IDE under an environment that tends to facilitate learning (in the fall semester of 2011), required an additional involvement in programming activities from novice undergraduates (compared to the implementation of fall 2010). This result is in line with previous research (e.g., [7], [9]). Besides helping students to enhance their understanding on the meaning of running programs, this stronger involvement among them may also lead to higher achievements [2], [4], [6], [17]–[19], [21]. Still, concerning those students who passed, the programming module final grade average of fall 2011 was not significantly higher than the final grade average of fall 2010. Nonetheless, the results seem to confirm that students' achievements may have been improved because students got highly involved in the programming tasks (e.g., for automatically animating increasingly difficult worked algorithmic solutions with Portugol IDE 2.3) throughout the module. This may have made the difference between students getting approval and dropping the programming module of PNM9703 course.

In line with previous empirical research [2]–[4], [19], [22], this study suggests that the successful integration of a program visualization tool (e.g., Portugol IDE 2.3) into an environment that tends to facilitate learning (like the one described here) requires lecturers to (i) pick a stable and easy to learn and use tool, (ii) introduce students to the tool in the beginning of the module, (iii) make sure that students use the tool throughout the module, giving them basic programming tasks (e.g., worked examples – constructed to avoid splitting students' attention between different sources of information or having them deal with redundant information [12], [25] – for novices to study, correct and/or complete), (iv) remind students (as needed) that they will be tested on the understanding of structure and function of pseudo-code sequences structurally identical to the ones trained in class, and (v) explicitly teach students how to use the tool and interpret its automatic visualizations (in the beginning and later on in the module, as needed), for instance, making them run basic algorithms (or worked examples) step-by-step at their own pace and giving students feedback on these algorithms and corresponding step-wise animations [15].

Future integrations of program visualization tools into CS introductory modules shall provide further insight into students' background and characteristics (e.g. demographics, programming experience, perceptions and attitudes towards CS, learning methods, and program visualization tools, and both team and individual work) and use of the tool. This data shall help confirm, in future studies, if the effects reported here on students' performance are from the tool or just 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: An Example of a Standard Programming Problem (Translated into English)

Write a program that computes the area of a triangle.

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] J. Kasurinen, M. Purmonen, and U. Nikula, "A study of visualization in introductory programming," in *Proc. 20th Annu. Meeting Psychology Programming Interest Group*, Lancaster, 2008.

[8] 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.

[9] E. Lahtinen, T. Ahoniemi, and A. Salo, "Effectiveness of integrating program visualizations to a programming course," in *Proc. 7th Baltic Sea Conf. Computing Education Research–88*, Koli National Park, 2007, pp. 195–198.

[10] C. Leão, G. Machado, R. Pereira, J. Paulo, and S. Teixeira, "Teaching differential equations: concepts and applications," in *Proc. International Conf. Engineering Education – New Challenges in Engineering Education and Research in the 21st Century*, Budapest, 2008.

[11] 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.

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

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

[14] 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.

[15] I. Moura, "Visualizing the execution of programming worked-out examples with Portugol," in *Lecture Notes in Engineering and Computer Science: Proc. World Congress on Engineering Vol I*, London, 2013, pp. 404–408.

[16] T. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. Velazquez-Iturbide, "Exploring the role of visualization and engagement in computer science education," *SIGCSE Bulletin*, vol. 35, no. 2, pp. 131–152.

[17] 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.

[18] 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.

[19] 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.

[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] J. van Merriënboer, J. Schuurman, M. de Croock, and F. Paas, “Redirecting learners’ attention during training: effects on cognitive load, transfer test performance and training efficiency,” *Learning and Instruction*, vol. 12, no. 1, pp. 11–37, 2002.
[26] M. Weimer, *Learner-centered teaching. Five key changes to practice*. San Francisco, CA: Jossey-Bass, 2002.
[27] S. Wiedenbeck, “Novice/expert differences in programming skills,” *International. Journal of Man-Machine Studies*, vol. 23, no. 4, pp. 383–390, 1985.

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)