Learning to Program with ProGuide

Cristiana M. Areias[1], António J. Mendes[2] and Anabela J. Gomes[3]

Abstract - **Problem solving and solution planning are probably the most difficult skills that novice programming students must acquire. When confronted with a programming problem many of them fail to create a solution proposal, even if it is not completely correct. When that happens many of them loose motivation and consequently stop working. In this paper we present ProGuide, a dialogue based tool to support weaker students to create basic programs. In ProGuide students are stimulated and guided through a text-based dialogue. The tool encourages students providing hints, questions, similar examples and so on, to help students reach the problem solution. We believe that this tool can help novice programming students, especially those that have more difficulties.**

Index Terms - Educational technology, Problem solving, Programming teaching and learning.

INTRODUCTION

Programming courses usually appear in the beginning of computer science and engineering curricula. This happens because programming skills are required in several other courses and students are expected to be able to program [1]. However, computer science educators know that initial programming courses often present high drop out and failure rates. In literature it is also possible to find many references to this major educational problem and its causes. As Carter and Jenkins say, it is common for student to approach their final year project determined to avoid programming at all costs, probably because they either cannot program or believe that they can not [2]. Learning to program is recognized to be difficult, since students must develop good problem solving skills, learn to express themselves in algorithmic terms and use programming languages that are often artificial for many of them [3].

It is possible to find in literature several discussions about the reasons that contribute to learning difficulties [4, 5]. Student mathematics and science backgrounds, student motivation, class sizes and heterogeneous groups of students in class, programming language syntaxes are among the most cited. We believe that the main difficulty for most students is to compose and coordinate available instructions to create the components of a program [6, 7]. Many times students understand basic programming constructs, but are unable to use them to create coherent programs that solve problems.

We believe that learning to program is essentially learning to solve problems algorithmically and the programming language should be just a way to express the solution. Programming knowledge cannot be directly transmitted from teacher to students. On the contrary, students must actively acquire that knowledge [8, 9], which means that practice of programming is a fundamental activity for novice students. However, many students find many difficulties in this initial learning phase, and many are not able to create solutions to simple problems. This creates conditions for students to lose motivation and give up trying, leading to drop out or failure.

As an answer for students learning difficulties, many researchers have proposed tools and approaches to support programming teaching and learning. Micro worlds, like Karel Robot [10,11] and Alice [12,13], try to introduce basic programming constructs through a familiar environment, where it is possible to use such constructs to control movements and other behaviors of some familiar entity (like a robot). Controlled development environments have also been proposed, so that students can develop programming skills in environments less complex than professional tools. THETHIS [14], X-Compiler [15], DrScheme [16] and BlueJ [17,18] are examples of such tools. Animation/simulation educational tools, such as Interactive Data Structure Visualization [19], the Programming Education System Based on Program Animation [20], SICAS [21], EROSI [22], JAWAA [23], JHAVÉ [24], Jeliot 2000 [25], OOP-Anim [26] and Raptor [27], try to help students to better understand programs through the utilization of graphical representations. Another know approach are tools like Lisp Tutor [28], C-Tutor [29], DISCOVER [30], and ELM-ART [31], that use artificial intelligence techniques to promote individualized learning. Some of those tools allow students to simulate their own programs, trying to help them to find and correct errors and misconceptions. Although, in our view, this process of error finding and correction supported by program simulation tools can be extremely valuable in educational terms, weaker students can't take advantage of it, simply because they aren't able to create a first solution proposal that may be simulated and improved. Also, the majority of those tools emphasize programming language features rather than problem solving skills and demand knowledge of a specific programming language, which weaker students often don't have. We developed ProGuide as an attempt to help those students to create their first solutions, hopping that after this contributes to develop their

[1] Cristiana M. A. Areias, Department of Informatics Engineering and Systems, Polytechnic Institute of Coimbra and CISUC - Department of Informatics Engineering, cris@isec.pt
[2] António José Mendes, CISUC - Department of Informatics Engineering, University of Coimbra, toze@dei.uc.pt
[3] Anabela J. Gomes, Department of Informatics Engineering and Systems, Polytechnic Institute of Coimbra and CISUC - Department of Informatics Engineering, anabela@isec.pt

abilities, allowing them to start creating solutions to simple programming problems.

ProGuide is a dialogue based tool to support weaker students to create basic algorithms. It includes a previously developed algorithm simulation tool, SICAS [21], and a dialogue based tool that interacts with students during algorithm development. The idea is to help students to create a first solution that can then be improved. In ProGuide students are stimulated and guided through a dialogue. The tool tries to encourage students, providing hints, questions, and similar examples. This interaction should make students reflect and step by step construct their solutions.

In next section we will describe the ProGuide environment. The third section includes an utilization example that may help clarify how the tool works. Finally, we present some conclusions and ideas for future work.

PROGUIDE ENVIRONMENT

ProGuide main goal is to help to reduce the difficulties that weaker programming students show in the initial learning stages. When students start programming they must actively construct knowledge assisted by guidance from teachers and feedback from others students [9]. Probably the most effective way to help those students would be to provide human tutoring that could assist each student during problem solving, giving support in their reasoning and immediate feedback on the programs they create [32]. However, this is usually not possible, due to the large number of students in classes and the limited number of available teaching staff.

To achieve its goal, ProGuide includes an algorithm editor/simulator and a dialogue based tool that interacts with students during algorithm development. ProGuide has internal structures to store information about problems and its solutions. It uses that information to interact with the student when a particular problem is proposed to him/her.

ProGuide interface is divided in tree sections as presented in Figure 1. The statement of the exercise on top left side, the tutoring and communication on center left side and finally the editor/simulator on the right side.

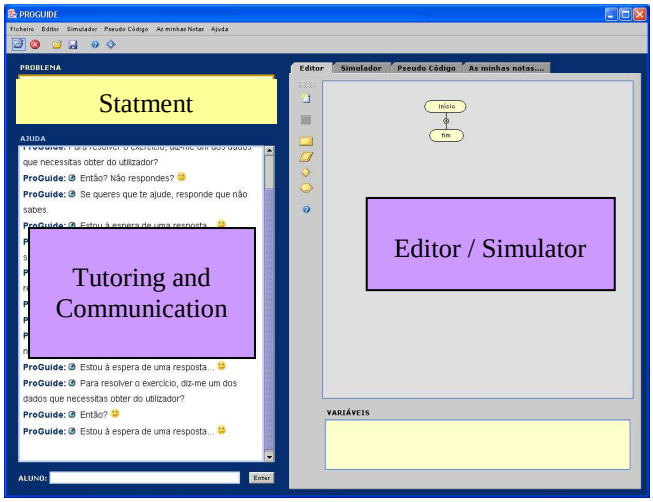

FIGURE 1
PROGUIDE ENVIRONMENT.

In the next subsections we will describe ProGuide main features.

I. The Editor and Simulator

The editor/simulator was inspired in SICAS, a tool developed previously in our research group. SICAS [21] is an educational environment designed to support the learning of basic concepts of procedural programming. It essentially allows the design and animated simulation of algorithms expressed by flowcharts. Like SICAS, ProGuide editor and simulator is a user-friendly iconic space that supports the design of algorithms using flowcharts. The option to use flowcharts was taken because we think this representation is more appellative, simpler and probably less prone to errors [33]. Another useful characteristic of flowcharts is their programming language independence, which allows ProGuide utilization in courses that use any procedural programming language.

As mentioned before, ProGuide aims to help students in their initial learning stage. The problems proposed to students at this stage are usually simple and require only basic programming constructs. That is why the editor only supports input/output, attribution, repetition and selection structures. When the student inserts one of those constructs in the flowchart, ProGuide verifies if it is correctly positioned in the flowchart, if its details are correct (for example the condition in a selection instruction), and if it makes sense to solve the proposed problem. This last verification is very important and is made using information given by the teacher during problem specification. This means that the introduction of new problems in the environment is complex, as the teacher must give enough information about the characteristics of the expected solution (for example, saying it must have a repetition with a selection inside) and also some input data and corresponding outputs that ProGuide will use to verify if the student solution works as expected.

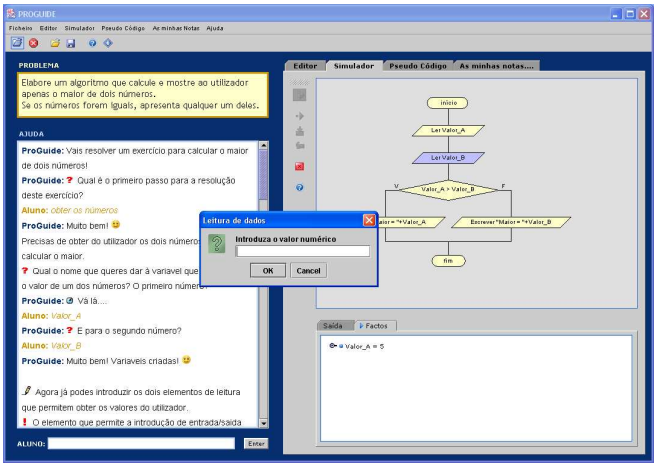

FIGURE 2
ALGORITHM SIMULATION.

Whenever the student wants, she/he can simulate the solution through the flowchart animation. It is possible to analyze the algorithm behavior, eventually identifying errors that may have to be corrected. During animation, the

flowchart element being executed is highlighted and the student can see the variable's current values and the output produced by the algorithm (figure 2).

II. Tutoring and Communication

The communication that ProGuide establishes with the students tries to help, encourage and guide them during algorithm design. This is made through hints, examples and questions designed to help students thinking correctly about the problem and its solution. Questions used are generally simple, but they may guide student's reasoning about the problem. For example, we use questions like:

- Which variables are known and unknown in the problem?
- Which information can be obtained from the problem description?
- Which data must be received from the user?
- What must be calculated?
- What must be presented to the user?
- Which instructions must be inserted in each step?

We believe that this kind of guidance can be useful to solve a particular problem, but also to develop problem solving habits that the students can use in other situations.

Communication with students is made using a subset of Portuguese natural language. Natural language processing is complex and difficult, so it possible that students give unexpected answers. In this case, ProGuide answers with "Sorry, I don't understand" or "Sorry, we aren't in the same context", inviting the student to give an alternative input. To reach our objectives it is enough to use a small subset of the Portuguese language, and there is no need to support very complicated statements.

When interacting with the student ProGuide frequently puts questions and waits for answers. Those questions have associated timeouts that trigger ProGuide when the students don't answer in the time given. In such cases, depending on the exact situation, ProGuide may:

- Repeat the question;
- Call the student with "Are you there?" or "Don't you answer?"
- Ask the student if she/he wants to see the solution to a similar problem;
- Present the answer and go to the next step.

III. Theoretical Information

Sometimes when we see students trying to solve a problem we conclude that they don't know basic programming aspects. When that happens it may be useful to ask students to read some information about those aspects (for example, repetition structures) and see other problems that are solved using those same aspects. ProGuide includes information and examples about concepts like variables, input/output, selections and repetitions. For each of them there are some texts (figure 3 shows an example about repetitions) and examples (figure 4). An interesting aspect is that the examples provided are flowcharts that can also be simulated in the environment, allowing a better student understanding.

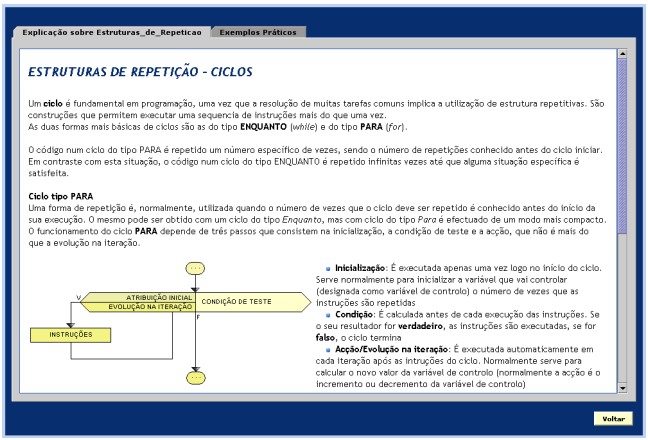

FIGURE 3
HELP ABOUT REPETITIONS

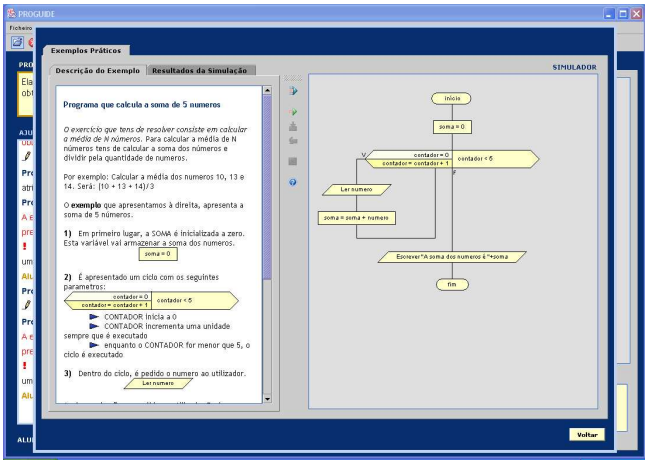

FIGURE 4
EXAMPLES WITH REPETITIONS

PROGUIDE UTILIZATION EXAMPLE

ProGuide includes several typical programming learning problems, such as "Calculate a rectangle area", "Determine the bigger of two numbers" or "Calculate the average of N values given by the user". As an example of ProGuide functioning, we will describe a small example, considering that a student has to create an algorithm to determine the bigger of two numbers. This is a very simple problem, often used to introduce the selection structure.

To solve that problem, the student must follow the following partial steps:

- Receive the numbers, which implies that she/he defines two variables to store them;
- Compare the numbers;
- Print the bigger number;

This means that the flowchart must have two input elements, one selection structure and an output element. As mentioned before, ProGuide stores information about the solution characteristics and the dialogues that can be used to help students reach that solution.

In this case, communication is established with a question: "Which is the first step to solve the problem?" Of course, ProGuide expects an answer like "Get the numbers".

If the answer is not the expected one, ProGuide tries to guide the student to conclude that the first step is to get the numbers from the user. As example, if the student answers "Compute de bigger one", ProGuide answers can be: "Yes! But to compute de bigger one what do you need to get from the user?", or "Yes! But you need to know the values, right?".

After the initial dialogue, the student must declare two variables. If the student takes too long or if she/he inserts an impossible/incorrect variable name, ProGuide informs her/him about the error and/or suggests that the student reads some more information about variables. To conclude this initial stage, the student must insert in the editor two input elements, one for each number to be read. ProGuide will not allow the dialogue to go to a further step before those elements are inserted correctly.

The following step is to compare the values to determine the bigger number. So, it will be necessary to insert a conditional structure in the flowchart. At this stage, ProGuide expects that the student inserts the necessary component in the flowchart. If the student fails, or the timeout is triggered, ProGuide behavior can be: present examples of algorithms where the selection structure is used in analogous situations (for example to determine if a student is approved given his grade), or insert some commentaries, hints, and question to help the student to think, like:

- "One of the most important parts of programming is controlling which statement will execute next. As programmer you can use control structures to determine the order in which your program statements are executed, the number of times that statements are executed, and whether or not statements are executed at all. Which control structures are needed to do that step where you must compare two numbers? Selection or Repetition?"
- "Remember, if the first number is the bigger one the output will be that number else it will be the other number. Which control structure you need to use to solve your problem? Repetition or Selection?"
- "When you need to do some actions based upon a decision you must insert a selection structure. A selection controls if some instructions are executed or not, or which of two groups of instructions are executed. For example: check if a number is negative before doing a square root; if true the program finishes else the program continues. So, go on to solve your problem… insert the correct structure."
- "You must compare the values to compute the bigger one…."
- "If you need to repeat a block of instructions several times, you need a repetition. If you need to do some actions based on a decision you must insert a selection. Insert the correct element in the flowchart…."

Similarly, in other problems, it may be necessary to lead the student to conclude that she/he needs to use repetition structures. In that case ProGuide uses hints/questions like:

- "A repetition structure controls how many times a block of instructions is executed. Do you need this kind of structure in your problem?"
- "Often you need to execute some instructions while a condition occurs. You cannot know in advance how many times you will need to execute the instructions, so you cannot simply copy them a specific number of times. But, if you knew how many times the instructions should be executed, copying them that many times is not a good idea… Maybe it is better to use a repetition structure"
- Or, in the context of "average of N values" problem, "Imagine that the user wants to compute the average of 5 numbers. To read those numbers, how many input elements should exist in the flowchart?", "But now, imagine the user wants to compute the average of 100 numbers. The program would need 100 input elements...", "And if we don't previously know how many numbers the user wants? If we only know that during program execution?"

Returning to the bigger of two numbers problem, after recognizing the need for a selection, the student has to insert the corresponding element in the flowchart. If the student inserts some other element (a repetition for example), or the selection element is introduced in a wrong position, ProGuide presents a suitable message, highlights the flowchart, and suggest the correction. A similar situation happens when the student inserts the correct element, but gives a wrong selection condition, as shown in figure 5.

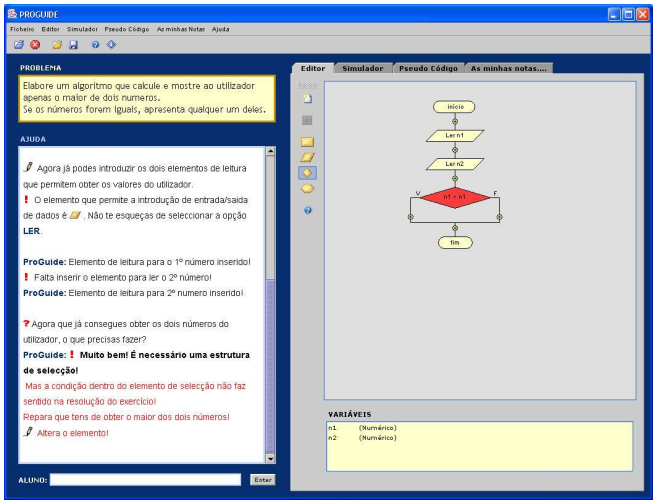

FIGURE 5
ERROR IN FLOWCHART

To conclude problem solution, ProGuide simply checks if the student inserts the output element to print the bigger number and advices the student to simulate the solution to check if it works as expected. To facilitate this, ProGuide provides some input values as test data, so that students do not test only with some values, but also with values that may cause errors if the program is not properly created.

CONCLUSIONS

Students must do a lot of practical work to learn programming. However, many of them find it difficult, loose motivation and consequently do not develop programming

abilities as expected. ProGuide is a dialogue based tool to support the construction of algorithms to solve some commonly proposed exercises. Using ProGuide, students learn and actively develop their problem solving skills, answering questions, and getting feedback for their actions. The main idea is to support weaker students that many times are not able to devise a solution to simple problems. Algorithm simulation and animation available in ProGuide is also a factor that may contribute to student motivation and learning. When they are able to propose a solution, the environment helps them to analyze it, and detect eventual errors that exist.

The introduction of new problems in ProGuide is not simple in the current version. All information to guide student during the dialogue must be explicitly created as it is specific to each problem. To reduce this problem we plan to create a user-friendly tool to help in that work.

ProGuide was not yet evaluated by students, but the preliminary evaluation done by some programming teachers was encouraging. We believe that ProGuide can help novice programming students, especially those with more difficulties.

REFERENCES

[1] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, , "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students", in *Proc. of 6th Annu. Conf. on Innovation and Technology in Computer Science Education*, 2001, pp. 125-180.

[2] J. Carter and T. Jenkins, "Gender and programming: What's going on?", in *Proc. of 4th Annu. Conf. on Innovation and Technology in Computer Science Education*, 1999, pp. 1-4.

[3] R. Moser, "A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it", *ACM SIGCSE Bulletin*, vol. 29, nº 3, pp. 114-116, 1997.

[4] N. Pillay, "Developing intelligent programming tutors for novice programmers", *ACM SIGCSE Bulletin*, vol. 35, nº 2, pp. 78-82, 2003.

[5] N. Pillay and V. Jugoo, "An Investigation into Student characteristics Affecting Novice Programming Performance", *ACM SIGCSE Bulletin*, vol. 37, nº 4, pp.107-110, 2005.

[6] R. E. Mayer, "The Psychology of how novices learn computer programming", *Computer Surveys*, vol. 13,nº 1, pp. 121-141, 1981.

[7] J. C. Spohrer and E. Soloway, "Putting it all together is hard for novice programmers", in *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics*, 1985, pp. 728-735.

[8] I. Boada, J. Soler, F. Prados, and J. Poch, "A teaching/learning support tool for introductory programming courses", in *Proc. of 5th Int. Conf. on Information Technology Based Higher Education and Training*, 2004, pp. 604-609.

[9] M. Ben-Ari, "Constructivism in computer science education", *Journal of Computers in Mathematics & Science Teaching*, vol. 20, nº 1, pp. 45-73, 2001.

[10] R. E. Pattis, *Karel the Robot: A Gentle Introduction to the Art of Programming*, 2nd ed., John Wiley & Sons, 1994.

[11] D. Buck and D. J. Stucki, "JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum", *ACM SIGCSE Bulletin*, vol. 33, nº 1, pp. 16-20, 2001.

[12] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts", *Journal of Computing in Small Colleges*, vol. 15, nº 5, pp. 107-116, 2000.

[13] C. Kelleher, D. Cosgrove, D. Culyba, C. Forlines, J. Pratt, and R. Pausch, "Alice2: Programming without Syntax Errors", *User Interface Software and Technology*, 2002.

[14] S. N. Freund and E. S. Roberts, "THETIS: An Ansi C programming environment designed for introductory use", *ACM SIGCSE Bulletin*, vol. 28, nº 1, pp. 300-304, 1996.

[15] G. Evangelidis, V. Dagdilelis, M. Satratzemi, and V. Efopoulos, "X-Compiler: Yet Another Integrated Novice Programming Environment", in *Proc. of 2nd IEEE Int. Conf. on Advanced Learning Technologies*, 2001, pp. 166-169.

[16] R. B. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen, "DrScheme: A programming environment for Scheme", *Journal of Functional Programming*, vol. 12, nº 2, pp. 159–182, 2002.

[17] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg, "The BlueJ system and its pedagogy", *Journal of Computer Science Education*, vol. 12, nº4, pp. 249-268, 2003.

[18] K. Van Haaster and D. Hagan, "Teaching and learning with BlueJ: an Evaluation of a Pedagogical Tool", in *Proc. of the Information Science and Information Technology Education Joint Conf.*, 2004, pp. 455-470.

[19] D. J. Jarc and M. B. Feldman, "An empirical study of web-based algorithm animation courseware in an Ada data structure course", in *Proc. of Annu. ACM SIGAda Int. Conf. on Ada*, 1998, pp. 68-74.

[20] Y. Miyadera, N. Huang, and S. Yokoyama, "A programming language education system based on program animation", in *Proc. of Education Uses of Information and Communication Technologies - World Computer Congress*, 2000, pp. 258-261.

[21] A. Gomes and A. J. Mendes, "Suporte à aprendizagem da programação com o ambiente SICAS", in *Proc. of V Congresso Ibero-Americano de Informática Educativa*, 2000.

[22] C. E. George, "EROSI—Visualizing recursion and discovering new errors", in *Proc. of 31st SIGCSE Technical Symposium on Computer Science Education*, 2000, pp. 305-309.

[23] S. H. Rodger, "Using hands-on visualizations to teach computer science from beginning courses to advanced courses", in *Proc. of the 2nd Program Visualization Workshop*, 2002, pp. 103-112.

[24] T. L. Naps, "JHAVÉ: Supporting algorithm visualization", *IEEE Computer Graphics and Applications*, vol. 25, nº 5, pp. 49-55, 2005.

[25] R. B. Levy, M. Ben-Ari, and P. A. Uronen, "The Jeliot 2000 program animation system", *Computers & Education*, vol. 40, nº 1, pp. 15-21, 2003.

[26] M. Esteves and A. J. Mendes, "A Simulation Tool to Help Learning of Object Oriented Programming Basics",in *Proc. of 34th ASEE / IEEE Frontiers in Education Conf.*, 2004, pp. F4C7-12.

[27] M. C. Carlisle, T. Wilson, J. Humphries, and S. Hadfield, "RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving", in *Proc. of 36th SIGCSE Technical Symposium on Computer Science Education*, 2005, pp. 176-180.

[28] J. Anderson and B. Reiser, "The LISP Tutor", *Byte*, vol. 10, nº 4, pp. 159-175, 1985.

[29] J. S. Song, S. H. Hahn, K. Y. Tak, and J. H. Kim, "An intelligent tutoring system for introductory C language course", *Computers & Education*, vol. 28, nº 2, pp.93-102, 1997.

[30] H. A. 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.

[31] G. Weber and P. Brusilovsky, "ELM-ART: An adaptive versatile system for web-based instruction", *Int. Journal of Artificial Intelligence in Education*, vol. 12, pp. 351-384, 2001.

[32] E. Hash and J. Zachary, "Automated Feedback on Programs Means Students Need Less Help From Teachers", *ACM SIGCSE Bulletin*, vol. 33, nº1, pp. 55-59, 2001.

[33] D. Scanlan, "Structured Flowcharts Outperform Pseudocode: An Experimental Comparison", *IEEE Software*, vol. 6, nº 5, pp. 28-36, 1989.

[34] A. Calloni and J. Bagert, "Iconic Programming Proves Effective for Teaching the First Year Programming Sequence", in *Proc. of 28th SIGCSE Technical Symposium on Computer Science Education*, 1997, pp. 262-266.