Computer-supported Techniques to Increase Students Engagement in Programming

Paula Correia Tavares, Pedro Rangel Henriques and Elsa Ferreira Gomes

Keywords: Programming, Learning, Students, Animation, Automatic Evaluation, Feedback.

Abstract: One of the main reasons that justify the student's failure in (introductory) programming courses is the lack of motivation that impacts on the knowledge acquisition process, affecting learning results. As soon as students face the difficulties concerning the development of algorithms or the coding in a programming language, they give up and do not try harder to solve other problems; they think it is a demanding activity and feel frustrated. In this paper we describe in detail an experiment conducted to verify the effectiveness, in terms of the increase in motivation and in knowledge acquisition, of combining program Animation tools with the immediate feedback provided by Automatic Evaluations Systems.

1 INTRODUCTION

Learning to programme is a complex task that poses significant challenges. Students face different kinds of difficulties at various levels and the traditional teaching/learning methods are no longer effective resulting in a high rate of failures (Hundhausen and Douglas, 2000). Such students' main difficulties are (Proulx, 2000):

- Understanding the problem due to their unfamiliarity with the subject or due to the inability to interpret the problem statement (identify its meaning);
- Thinking in a logic way to decompose the given problem into successively smaller parts and to write the correct algorithm (sequence of unambiguous and elementary operations) to solve it;
- Learning the language syntax and semantics.

The difficulties above, identified in learning programming, led to the creation of languages and development of environments that smooth the designing of algorithms, or the writing and the analysing of programs. However, as far as we know, that problem is not yet satisfactory solved.

New teaching/learning approaches must be devised. The resort to computer-supported education specially tailored to programming activities shall be explored. Animation can help students on the analysis and understanding of given programs, and it can also guide the development of new ones. For this reason, several authors (as discussed in section III) have researched the pedagogical effectiveness of program visualization and animation, and developed supporting tools.

According to our experience, it is crucial to give students the opportunity to practice solving programming exercises by themselves since the beginning of the course. Receiving feedback is essential for knowledge acquisition (Verdú et al., 2011). Immediate feedback is important to give the student an assessment of his ongoing work, indicating whether the result is right or wrong and, if possible, explaining the error causes and recovery. New tools arose (especially in the area of programming contests) to allow for the submission of solutions (programs developed by the students) to the exercises proposed by the teacher and to assess them, returning immediately information about the submitted answer (Verdú et al., 2011). These tools can be incorporated into teaching activities, allowing students to test their work getting immediate feedback. Automatic Assessment or Evaluation systems, as they are called, significantly improve students' performance. We believe that this approach can increase their engagement and consequently improve their academic success (Joy et al., 2005).

The goal of the research work, under which this paper appears, is to identify the difficulties that actually arise in the process of teaching/learning programming, and to suggest different approaches, supported on computer resources, to overcome the main difficulties. So, the expected outcome of this project is a set of strategies, focused on the motivation of learners, to improve the success of programming courses. In order to increase the motivation and self-confidence of students of introductory programming courses, these strategies will be based on the use of computers and computer applications that can increase the engagement of students in comprehension and development tasks. For that purpose it is essential to face students with challenging problems to solve providing them immediate feedback about their solutions. The simulation of the program execution with a visual interface is also relevant to attract student's attention and interest.

In this paper we discuss how two strategies for teaching programming, animation (see Section 3) and automatic assessment (see Section 4), can be combined into a new pedagogical practice (our proposal is introduced in Section 5). We also discuss the lessons learned from a first experiment conducted with students in a classroom (Section 6) to sketch future tests in order to refine our proposal. Before going into details, we briefly review in section 2 the process of learn how to programme learning programming.

2 LEARNING PROGRAMMING

Two different concepts that are usually misunderstood by students are: learning programming and learning the syntax and the semantics of a programming language. Programming is, first of all, to outline strategies in order to solve problems, regardless of the language used. In fact, this task involves several steps that go from the understanding of the work proposal to the test of the program, passing through the algorithm development and its codification. In (ACM/IEEE, 2013) there is an interesting discussion concerning the "programming focus" (read in this context, *practical*, or *coding-oriented*) character of the majority of introductory Computer Science courses. The authors of the referred guidelines (followed by academies world-wide) however alert "Whether or not programming is the primary focus on their first course, it is important that students perceive computer science as only learning the specifics of particular programming languages. Care must be taken to emphasize the more general concepts in computing within the context of learning how to program".

Although we believe that the codification is not the main difficulty, previous studies (Gomes, 2010), concluded that the adopted programming paradigm and the language used have a huge impact in the learning process and consequently in the task performance. However it does not exist yet any agreement among the computing community about the best paradigm or the most adequate language; opinions diverge! Citing again the same ACM/IEEE Computer Science curricula guideline, above referred (ACM/IEEE, 2013) the authors sustain a nice discussion on "Programming Paradigm and Choice of Language", emphasizing the inexistence of consensus among academics and recommending the presentation of alternative programming paradigms "to give students a greater appreciation of the diverse perspectives in programming,…"

Another important evidence that must be taken into account is that learning how to program is an iterative process. The solution to a complex problem can be obtained in a successive steps solving simpler problems and enriching the previous solutions. Composition of simpler solutions is also many times used to solve bigger problems. These techniques of enrichment and composition should be considered in the planning of teaching activities, leading to the proposal of problems in incremental steps of increasing complexity.

We also believe that it is only possible to learn how to program by programming. Following this approach, students can understand and acquire problem-solving strategies. Therefore, it is obvious that an active behaviour by the student, instead of a passive one, leads to an improvement of his ability to solve the proposed problems. However, teachers realize that in most cases, when students are requested to solve a particular problem, they are not able to start the task, neither on paper nor in the computer. Even when they break this initial inertia, they often become discouraged and give up easily as soon as they face the first hurdle (Proulx, 2000). In the opinion of some colleagues this statement is not true when teaching good (top level) students; however, they agree that this actually happens with the majority of normal students.

In this context and considering the facts above, we will discuss in the next two sections, the animation strategy and the tools supporting it, and the importance of feedback in the teaching-learning process. In this perspective, we will analyse the impact of tools for automatic evaluation of programs

that can be integrated in teaching process.

3 ANIMATION

The animation tools provide a visual metaphor that significantly helps the understanding of complex concepts. Therefore, these tools allow the students to find the dynamics of hard to understand but extremely important processes. In this way, the student is stimulated to progress in his activity (Hundhausen et al., 2002).

Several authors have been concerned about the use of graphic interfaces that enable a way of communication between the user and the computer not restricted to a textual form (Hansen et al., 1999) (Stasko and Kehoe, 1996) (Hundhausen and Douglas, 2000).

Aiming to enhance the learning process, many educators and computer scientists have been working on animation, visualization and simulation systems (computational programs). The great motivation is to appeal to the human visual system potential.

The key question is how to apply these methods in order to help students to deal with complex concepts.

Many researches (Brown and Sedgewick, 1985) (Korhonen, 2003), (Kerren and Stasko, 2002) have been working to identify the rules that should be followed while designing and creating visualizations and animations effective for teaching. As computer programs can be hard to understand when presented in a textual format, it is expected that a better comprehension could be achieved with an animated graphic format (Pereira, 2002).

An animation is a natural approach of expressing behaviours. Particularly, the animation of an algorithm is a dynamic visualization of their main abstraction. So, its importance lies on the ability to describe the algorithm logical essence.

When inspecting the control and data of a program to understand its behaviour, we have two big choices: do it during code execution (debugging), or simulate the execution in another environment (Pereira, 2002). For teaching purposes we believe that the second approach is clearly the most interesting (Stasko and Kehoe, 1996).

Several authors have worked on this problem. They develop less complex and appealing environments than the professional environments, with important features for novice programmers. These systems allow understanding important aspects in programming through the animation of pseudo-code, flowcharts, or programs written in specific or in a general programming languages (such as Pascal, C, Java, and others). The most interesting and appealing are those that allow students to introduce and simulate their own algorithms and programs. The animation based on simulation allows the production of dynamic visualizations of a program and help student comprehension at his own pace. In this context, there are several tools that try to introduce basic programming concepts through a familiar and pleasant environment in order to help students on learning to program. The following list shows some of the most well-known tools: BALSA (Saraiya, 2002), TANGO (Hughes and Buckley, 2004), Jeliot (Silva et al., 2009), Alma (Pereira and Henriques, 1999), SICAS (Mendes et al, 2004), OOP-Anim (Santos et al, 2010) (Esteves and Mendes, 2003), VILLE (Rajala et al., 2007), JIVE (Lessa et al., 2011). All of these tools are concerned with visualization or animation of programs written in traditional programming languages (C, Java, etc.).

To illustrate this idea, we show in Figure 1 a screenshots of Jeliot System.

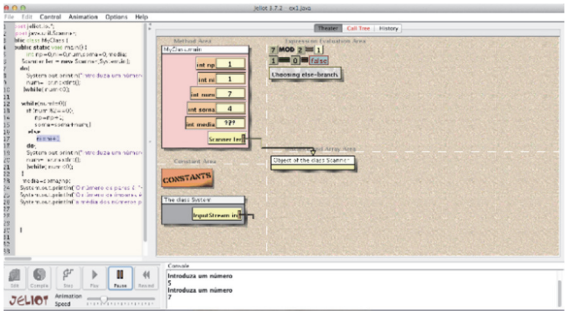

Figure 1: Jeliot interface - animating exercise A.

Figure 1 shows a moment in the animation of Exercise A (checking the condition: *is 'n' even?*) included in the experiment that will be introduced in Section 6. Figure 1 illustrates the step when the program checks if a value is an even number; for each exercise, the sequence of images of this kind (corresponding to the execution of each statement) produces the animation of the program under study, as desired.

4 AUTOMATIC EVALUATION

It is very important to give students the opportunity to practice and solve programming exercises by themselves. However, the maximum effectiveness of this approach requires the teacher's ability to review, mark and grade each solution written by students. Instant feedback is very important for the acquisition of knowledge. Independently of the particular

learning strategy, it motivates students.

However, in large classes and with few lecture hours, this approach is impractical. Individual feedback may consume too much teacher´s time with risk that students do not benefit from it in due time (Queirós and Leal, 2015).

To solve this problem, there are some online submission systems that support the automatic evaluation of programming problems (Queirós and Leal, 2012). Different studies show that these systems enable students to autonomously develop programming skills and significantly improve their performance (Verdú et al., 2011).

Since not all students are motivated in the same way, it is important to provide different learning environments: individual (traditional), collaborative (group work), competitive (contests), among others. The role of group in education and students tracking in collaborative environment is discussed in (Boas et al., 2013) (Fonte et al., 2014). In this paper the authors propose the use of Continuous Integration techniques, usual in Agile Development (Elliott et al., 2015) (Awad, 2005), to support incremental group work providing immediate feedback to students and teachers. Taking advantage of the human spirit of competition, competitive learning increases commitment and leads to a greater involvement of students in practical activities. So, competitions with automatic evaluation are becoming important for the practice of programming. However, differences in motivation and feelings between losers and winners can exist. These negative effects can be minimized through different practices, such focusing on learning and fun rather than in the competition by itself (Verdú et al., 2011).

New tools have emerged to facilitate and enable their use in teaching activities, allowing students to incorporate tests in their work. These tools increase the level of satisfaction and motivation of students. According to teachers and students, feedback should be provided and detailed as quickly as possible. These tools do not replace the teacher, but provide help and increase the value of time in the classroom. Teachers should be able to select the problems they intend to present to the students according to their level of difficulty (Verdú et al., 2011).

With suitable software tools, correctness of the program can be determined with an indication of the quality according to a set of metrics. It is not easy to find a unique approach to the problem of assessment of programming works. Different teachers can adopt different strategies, depending on their specific goals and objectives of the course, especially of their own style and preferences (Joy et al., 2005). So, the problem is related to the resources required to manage the evaluation of practical exercises. The students receive accurate feedback at the right time to the benefit of their learning.

Most of the tools available for this purpose include a submission subsystem to upload the student's works and another one for their automatic evaluation. This is adequate for an initial learning where knowledge and understanding are being tested. The final goal is to provide new learning strategies to motivate students and make programming more accessible and an attractive challenge.

Boss (Heng et al., 2005), Mooshak (Leal and Silva, 2008) and EduJudge (Verdú et al., 2011) are examples of such tools. They compare the output produced by the submitted answers (programs) against the expected output (repeating the verification for a set of input/output test cases) and produce a grading, but, at the same time, these systems help the teacher to involve students through precise and rapid feedback.

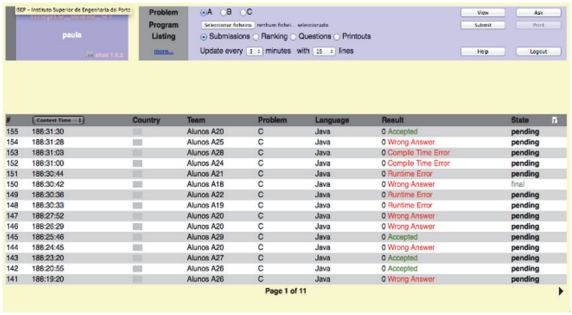

Figure 2: Mooshak System Interface.

Figure 2 shows a Mooshak screenshot illustrating the simplicity and easy of its interface. In addition to the feedback of exercises, shown in the central window, it is possible to see, on the top window, the different options offered to the students (exercise selection, submission, visualization of the results, etc.). This image was collected during the experiment that will be presented in section 6.

5 COMBINING ANIMATION AND AUTOMATIC EVALUATION

In this section we introduce our proposal aimed at improving the students' engagement and motivation. For that purpose we sketched an approach based on the following principles: To provide means for an easier understanding of programs, and so help on the

writing of new ones; to make students increase their ability to regularly practice regularly programming, since the first day, obtaining immediate feedback.

To attain the above mentioned objectives and regarding the currently available computing resources, we propose to combine Animation and Automatic Evaluation techniques. To be more specific we introduce in the sequel a summary of this proposal.

Step 1: We suggest that for each topic to teach, the teacher prepare three similar exercises.

Step 2: For the first exercise, the teacher shall analyse with the students the problem statement, and then ask them to solve the problem and test the solution produced using the Automatic Evaluation System, AES, selected. The teacher can also discuss with each student the feedback received.

Step 3: Finally, the teacher provides his solution for the exercise and the student shall use the selected Animation System, to animate the execution of the given program in order to carefully analyse and understand the correct solution and its behaviour.

Step 4: Repeat steps 2 and 3 for the remaining exercises.

This approach assumes that the teacher selects a powerful Animation tool, easy to use, and chooses an AES that is user-friendly and returns a feedback as complete as possible (with a diagnosis for the errors found). It is also desirable that AES comments the code quality. For our experiment, we chose Jeliot and Mooshak.

6 EXPERIMENT AND DISCUSSION

In this section we describe a first experiment conducted with the following main objectives:

- to understand the behaviour of students facing a new and different situation;
- to observe if students are involved and motivate;
- to determine if students improve their performance in programming (solving problems).

We also discuss the results that we got out of it.

In our case, to teach the introductory topic "*sequential numeric processing; conditional and iterative control structures*" we wrote the three exercise statements below:

a) *Write a program to read a sequence of positive integers that ends with a zero (0). The program must compute the amount of odd and even numbers as well as the average (float) of the even values.*

b) *Write a program that, given a number M and a number N, both positive integers, reads N ages printing all ages greater than M. At the end, the program must compute and display the average (float) of the ages read.*

c) *Write a program that, given the temperatures (float) of 6 days of January (values between -50° and 50°), compute and print the maximum and minimum temperatures. Also classify the month as "cold" or "warm" as it had more days with negative temperatures or with positive temperatures (zero included). In case of equality consider that the month is "cold".*

After deciding the concrete tools to use, the topic of the experimental lesson, and the exercises to solve, it was necessary to write down a careful plan for the lesson, so that all the students enrolled could understand what they are asked to do and how should they proceed. For that purpose, we have sketched a detailed plan for the lesson composed of nine different small tasks (omitted for the sake of space).

This plan was drawn as a flow chart; it was printed and distributed to all the students.

Before starting the experience, Jelliot was installed in all the computers in the classroom, and Mooshak was configured and prepared in a server provided by the Department of Computing at FCUP in order to be accessed by the students via Internet (remember that Mooshak is a Web-based tool accessible online).

The experiment involved 28 students of 1st year engineering degree (not in computer science). The students were split in two classes. Each class was supervised by two teachers that have stayed all the time in the classroom to help students, and to observe carefully the session aiming at getting a precise track of the experiment.

Along the session, the time for each exercise and each phase (resolution and automatic evaluation with Mooshak; visualization/animation of the correct solution with Jeliot) was strictly controlled in order to guarantee that all the subtasks could be executed in the class duration (2 hours). The first half an hour was used to prepare students for the session; the flow chart was distributed and explained. The remaining time, 90 minutes, was divided into three equal parts; we have allocated half an hour to each exercises, 15 minutes to develop and test a solution and another 15 minutes to animate a correct solution.

Concerning the first objective, the teachers present in the room observed and reported that both sessions ran successfully; no incidents were recorded, and all the tasks planned were accomplished.

Regarding the second objective, once again the observers reported that all the students were completely engaged in finishing the activities propose.

In the next paragraphs we discuss the third objective.

At a first glance, and according to the figures collected from Mooshak and summarized in Table 1, we think that the behaviour of the students had actually changed along the class, and their productivity has increased (they slightly solved more easily the proposed exercises). As can be noticed in Table 1, the number of correct answers has increased.

Table 1: Summary of the experiment results.

	Ex. 1	Ex. 2	Ex. 3
Nº of correct answers	4	6	9
Nº of Wrong answers	12	11	14
Nº of compilations Errors	30	37	20
Total nº of submissions	46	54	43
Average of submissions	1,6	1,9	1,5
Correct answers after error	1	2	4
Correct answers at 1st sub.	1	4	4

In Table 1, the first line records the number of submissions for each exercise that were completely accepted by Mooshak, this is, that produced the expected output for all the given input values. Lines two and three record, the total number of submissions that were evaluated by Mooshak as Wrong (the output produced is not the expected one) or as Error (Compilation or Runtime error). Notice that a student can submit more than once until getting a correct solution. So the total of submissions shown in the fourth line is a measure of the students' activity, persistence, and the difficulty of the exercises. The third last lines present details about the submissions in order to refine the conclusions that can be inferred from fourth line.

So, we can observe in our experiment that the students solved the third exercise faster (small number of submissions) and with better results (total number of accepted submissions). It is important to emphasize that the last exercise was not easier than the previous. In our opinion this conclusions corroborates our hypothesis.

As a final comment, we notice that students showed a greater difficulty to solve the second exercise (according to the number of submissions) on account of the problem statement that was a bit more elaborate. Besides the numeric information displayed in Table 1, both teachers present in classroom also observed this evidence. Our comment is corroborated by students' answer to the questionnaire as shown in the next subsection. This conclusion also supports our hypothesis that students have problems to understand statements.

A lesson learned was that the resolution time was a bit restrictive. We believe that, if students had more time to solve the exercises, the results would be, on one hand, more successful, and on the other hand much more effective and motivating in terms of the learning activity. Namely, our observation told us that the animation phase would benefit if it were possible to allocate it more time.

6.1 Student Opinions

At the end of the experiment, each student answered to a short inquiry with three questions. Below we list the queries followed by a graphical representation of the responses distribution.

Q1- Have you more difficulty in the interpretation of the statement or in the coding of the exercises?

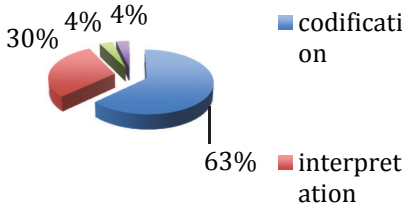

Figure 3: Student response to question number 1.

Most of the students answered that their greatest difficulty was in coding. However, 30% of students showed difficulty in understanding the statements. As mentioned earlier the interpretation of statements is an effective problem in programming (Figure 3).

Q2- Was Mooshak actually useful for your progress?

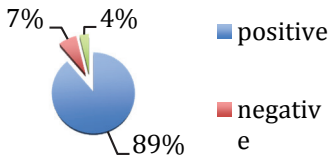

Figure 4: Student response to question number 2.

Mooshak was undoubtedly helpful for students (Figure 4). Most students reinforce that it was important to understand quickly whether the exercise

is correct or wrong. Students also refer the importance of displaying the error type, giving the possibility to correct it and submit the exercise again.

Q3- Do you think important the algorithm animation in Jeliot? In what ways?

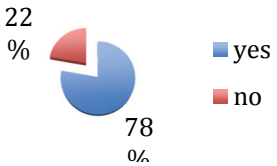

Figure 5: Student response to question number 3.

Most students consider important the animation offered by Jeliot (Figure 5). Students said that they got a better understanding of the algorithm and it was relevant because it explains well the problem solving it incremental and interactive way.

7 CONCLUSIONS

In this paper we have proposed an approach to improve the teaching/learning activity in introductory programming courses combining immediate feedback provided by Automatic Evaluation Systems with Program Animation Tools. To support our proposal, introduced in Section 5, we have designed and conducted an experiment that was described in Section 6.

As discussed in Section 6, the evolution of the students' behaviour along the two-hour lesson showed that the approach led them to better performance. On one hand, we notice that the number of the students with accepted submissions has increased. On the other hand, the number of trials increased and the number of compilation errors decreased. This means that the motivation of the students augmented while the basic errors decreased. Motivation was one of our main concerns.

The experience reported also allowed us to understand how to better conduct future tests. More flexibility in the time management during the lesson is one of the improvements that we want introduce. This means that we intend to propose the three exercises at the beginning and allow the students choose the time slice to use in each one; in this way, they can decide to explore deeper the animation.

To validate these conclusions, we think that it is necessary to repeat the experiment for other topics, like string processing and array processing, involving other student samples. These experiments are under preparation.

Another direction for future work is to compare the approach described against a variant of it that starts with Animation before students start their own resolution.

ACKNOWLEDGEMENTS

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013. We want to thank the valuable reviews and fruitful discussions with António Osório and Miguel Coimbra that really helped us to improve previous versions of this paper. We also acknowledge Maria João Varanda e Nuno Oliveira for their comments.

REFERENCES

ACM/IEEE, 2013. Computer Science Curricula 2013 -- *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, Final Report.

Awad, M., 2005. A Comparison between Agile and Traditional Software Development Methodologies. Submitted as partial fulfilment of the requirements for the Honours Programme of the School of Computer Science and software Engineering, *The University of Western* Australia.

Boas, I., Oliveira, N., Henriques, P., 2013. Agile development for education effectiveness improvement. *In Proceedings of the XV international symposium on computers in education (SIIE'2013)*. Viseu, Portugal.

Brown, M., Sedgewick, R., 1985. Techniques for Algorithm Animation. *IEEE SOFTWARE Vol 2(1),* pp 28-39.

Elliott, E., Fons, F., Randell, A., 2015. *Business Architecture and Agile Methodologies.* Business Architecture Guild, February 2015.

Esteves, M., Mendes, A., 2003. OOP-Anim, a system to support learning of basic object oriented programming concepts. *International Conference on Computer Systems and Technologies - CompSysTech'2003*.

Fonte, D., Boas, I., Oliveira, N., Cruz, D., Gançarski, A., Henriques, P., 2014. Partial Correctness and Continuous Integration in Computer Supported Education. *In Proceedings of the 6th International Conference on Computer Supported Education (CSEDU 2014)*, Volume 2. Barcelona, Spain.

Gomes, A., 2010. Difficulties of learning computer programming. Contributions to the understanding and resolution, Dificuldades de aprendizagem de programação de computadores: contributos para a sua compreensão e resolução. *Dissertação submetida à Universidade de Coimbra para obtenção do grau de "Doutor em Engenharia Informática".*

Hansen, S., Narayanan, N., Schrimpsher, D., 1999. Helping Learners Visualize and Comprehend Algorithms. *Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA'99)*.

Heng, P., Joy, M., Boyatt, R., Griffiths, N., 2005. Evaluation of the BOSS Online Submission and Assessment System.

http://www.uefs.br/erbase2004/documentos/weibase.

Hughes, C., Buckley, J., 2004. Evaluating Algorithm Animation for Concurrent Systems: AComprehension-Based Approach. *16th Workshop of the Psychology of Programming Interest Group.* Carlow, Ireland, April. In E. Dunican & T.R.G. Green (Eds). Proc. PPIG 16. pp. 193-205.

Hundhausen, C., Douglas, S., 2000. Using Visualizations to Learn Algorithms: Should Students Construct Their Own, or View an Expert's? *Proceedings 2000 IEEE International Symposium on Visual Languages IEEE Computer Society Press, Los Alamitos*.

Hundhausen, C., Douglas, S., Stasko, J., 2002. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of VisualLanguages and Computing* 13, pp. 259-290.

Joy, M., Griffiths, N., Boyatt, R., 2005. The BOSS Online Submission and Assessment System. *Journal on Educational Resources in Computing, Volume 5 Issue 3,* September 2005.

Kerren, A., Stasko, J., 2002. Chapter 1 "Algorithm Animation", Volume 2269, pp. 1-15.

Korhonen, A., 2003. Visual Algorithm Simulation. Dissertation for the degree of Doctor of Science in Technology. *At Helsinki University of Technology* (Espoo, Finland), November 2003.

Leal, J., Silva, F., 2008. Using Mooshak as a Competitive Learning Tool.

Lessa, D., Czyz, J., Jayaraman, B., 2011. JIVE: A Pedagogic Tool for Visualizing the Execution of Java Programs. *SIGCSE 2011 Dallas, Texas, USA.*

Mendes, A., Gomes, A., Marcelino, M., 2004. Evaluation and evolution of a Environment Support for Programming Learning, Avaliação e Evolução de um Ambiente de Suporte à Aprendizagem da Programação. *VII Congresso Iberoamericano de Informática Educativa*.

Pereira, M., 2002. Systematization of Programs Animation, Sistematização da Animação de Programas. Dissertação submetida à Universidade do Minho para obtenção do grau de doutor em Informática, ramo Tecnologia da Programação, December 2002.

Pereira, M., Henrriques, P., 1999. Made Algorithms Animation Systematic, Animação de Algoritmos tornada Sistemática. *In 1º Workshop Computação Gráfica, Multimédia e Ensino*. Leiria.

Proulx, V., 2000. Programming patterns and design patterns in the introductory computer science course. *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education,* pp.80-84. New York.

Queirós, R., Leal, J., 2012. Exercises Evaluation Systems - An Interoperability Survey. *In Proceedings of the 4th International Conference on Computer Supported Education (CSEDU), Volume 1, pp.83-90.* Porto.

Queirós, R., Leal, J., 2015. Ensemble: An Innovative Approach to Practice Computer Programming. In R. Queirós (Ed.), *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming* (pp. 173-201). Hershey, PA: Information Science.

Rajala, T., Jussi, M., Erkki, L., Salakoski, K., 2007. VILLE – A Language-Independent Program Visualization Tool. *Seventh Baltic Sea Conference on Computing Education Research* (Koli Calling 2007), Koli National Park, Finland, November 15-18.

Santos, Á., Gomes, A., Mendes, A., 2010. Integrating New Technologies and Existing Tools to Promote Programming Learning. Algorithms, Vol3, pp.183-196.

Saraiya, P., 2002. Effective Features of Algorithm Visualizations. Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University for *the degree of Master of Science In Computer Science*, July 2002.

Silva, M., D'Emery, R., Neto, J., Bezerra, Y., 2009. Programming structures: A Experiment with Jeliot,Estruturas de Programação: um Experimento com Jeliot. *IX Jornada de Ensino Pesquisa e Extensão (JEPEX) da UFRPE*.

Stasko, J., Kehoe, C., 1996. Using Animations to Learn about Algorithms: An Ethnographic Case Study. Technical Report GIT-GVU-96-20, September 1996.

Verdú, E., Regueras, L., Verdú, M., Leal, L., Castro, J., Queirós, Q., 2011. A distributed system for learning programming on-line. *Computers & Education* 58, pp. 1–10.

Xavier, G., Garcia, D., Silva, G., Santos, A., 2004. *Factors that Influencing Introductory Learning Programming,* Estudo dos Fatores que Influenciam a Aprendizagem Introdutória de Programação.