An environment to improve programming education

Conference Paper · January 2007

DOI: 10.1145/1330598.1330691 · Source: DBLP

2 authors:

Anabela Gomes
Instituto Politécnico de Coimbra
82 PUBLICATIONS **965** CITATIONS
SEE PROFILE

Antonio Jose Mendes
University of Coimbra
137 PUBLICATIONS **1,476** CITATIONS
SEE PROFILE

An environment to improve programming education

Anabela Gomes, António José Mendes

Abstract: *Computer Programming learning is a difficult process. Experience has demonstrated that many students find difficult to use programming languages to write programs to solve problems. Student failure and commonly expressed difficulties in programming disciplines suggest that traditional teaching approaches and study methods are not the most suitable for many students. There are several reasons that cause this learning problem, such as the lack of problem solving abilities that many students show. They don't know how to create algorithms, mainly because they don't know how to solve common problems. In this paper we describe several educational computer tools used successfully to support programming learning and we present a proposal for another computational system that may help reducing current problems.*

Key words: *algorithm animation, software visualization, educational technology, programming teaching and learning, learning styles*

INTRODUCTION

The high failure levels common in initial programming courses have been the investigation theme for many researchers. They try to find the causes for those problems and propose tools and teaching strategies that may help reducing them.

The main objective of initial programming courses is normally to develop student's problem solving abilities and teach them a programming language that may be used to express their solutions. It is important that student's develop programming abilities, so that they can create programs that solve real problems. However, experience has shown that many students have a huge difficulty to understand and apply certain abstract programming concepts when solving a problem. Many novice students even show difficulties to use basic concepts, like control structures, to create algorithms that solve concrete problems. These difficulties can be explained by different reasons [1, 2, 3], such as:

- Programming demands a high abstraction level;
- Programming needs a good level of both knowledge and practical problem solving techniques;
- Programming requires a very practical and intensive study, which is quite different from what is required in many other courses (more based in theoretical knowledge, implying extensive reading and some memorization);
- Usually teaching cannot be individualized, due to common classes size;
- Programming is mostly dynamic, but usually thought using static materials;
- Teachers' methodologies many times don't take into consideration the student's learning styles. Different students have different learning styles and can have several preferences in the way they learn;
- Programming languages have a very complex syntax with characteristics defined for professional use and not with pedagogical motivations.

In this paper we firstly present a literature review about several approaches to support programming learning. Later, a first specification of a new environment currently under development is presented.

PROGRAMMING LEARNING SUPPORT SYSTEMS

A literature review shows that many tools have been proposed to help solving programming learning difficulties. Many of those tools use animation and simulation techniques, trying to take advantage of the human visual system potential. When compared with static formats, we think animation can contribute to a better understanding of programming inherently dynamic concepts.

There are many visualization systems, some focusing on algorithms others on complete programs. Some focus on low level details (e.g. showing data structures and their evolution during a program), while others use a higher detail level, showing program behaviours, component relations and methodologies. Some systems just animate pre-defined programs and/or data structures, while others accept student's programs, allowing them to see how they work and, eventually, make the necessary corrections.

Some systems, like MRUDS - Multiple Representation for Understanding Data Structures [4], tried to go a step further, in this case using multiple visual representations to illustrate linear data structures, such as tables, stacks, queues and lists. Those representations are suitable for students with little or no knowledge about data structures.

Many other systems have been proposed, using visual representations or algorithm animations. BALSAII [5], VIP [6], Xtango [7], Jeliot 2000 [8] and Trackla [9] are known examples. Other proposals have been developed based on the belief that there are cognitive advantages resulting from the utilization of graphical metaphors [10]. Using this knowledge some new languages were created, like iconic programming languages (e.g. BACII [11] and BlueJ [12]), design languages like G2 [13] and web-based environments like Jhavé [14].

Artificial intelligence techniques to support programming teaching and learning have also been proposed, namely Intelligent Tutoring Systems, such as Lisp-Tutor [15], C-Tutor [16] and Cimel ITS [17].

Microworlds are another popular proposal, seriously influenced by the turtle graphics of LOGO [18]. Examples include "Karel the Robot" [19], Tortoise [20], TurtleGraph [21] and Alice [22].

The above mentioned systems were created mainly in the scope of academic works. However, some professional systems can also facilitate programming and the detection of programming errors. Modern IDEs are examples of such tools.

Acknowledging that there are many systems designed to support programming learning, it is relevant to ask why learning problems continue to be widely reported. It is also possible to ask why many of these tools are not widely used. The answer to these questions is not easy. Maybe some of them are not well known by teachers and students and possibly some are not well suited for students that mostly need support (those with deeper difficulties). We may also think that different approaches are still necessary, so that the computational environments can really help students, adapting themselves to student's preferences and needs. Our own work is an example of this situation. As we think that the main problem is the students' incapacity to create solutions to problems, in other words, to construct algorithms, we created a system called SICAS (a Portuguese acronym for Interactive System for Construction of Algorithms and its Simulation) [23]. This system focus essentially on algorithm development, allowing students not only to understand algorithms, but mostly to design, test, try, and correct their own algorithms.

However, in our opinion, there are also other factors that may contribute to student's difficulties, such as inadequate learning and studying methods, the students' previous difficulties in generic problem solving (and not only in programming), the inherent difficulty of programming and a negative connotation associated with the subject that has been growing lately among students. Based on all those factors, we are developing a new educational environment that will try to give a better support to the students.

We think that the major reason beyond student's difficulty to create algorithms is their low problem solving abilities, not only when facing programming problems, but also in several other domains. So, we believe that if those students can improve their generic problem solving abilities, that would have a positive impact in their programming competence. Thus, our system focus on the development of student's problem solving capacities, based on a constructivist approach of learning, where students learn by doing, trying and deducing, gradually constructing their own knowledge.

ENVIRONMENT DEFINITION

To define the environment characteristics it is important to make a correct diagnosis of student's difficulties and their causes. This will allow us to design features that may be useful to really help the students. To achieve this we are conducting some experiments to more precisely identify the causes for student's difficulties. Those experiments include activities, such as generic problem solving, algorithm development, detecting algorithms logical errors, other subject prerequisites (for example mathematical concepts used in problems), and student's misconceptions about programming techniques, among others.

To complement this study, we are also analyzing common teaching and learning approaches. To do so, we collect strategies used by programming teachers and analyze how they can contribute to the development of the students' problem solving capacity. Additionally, we also intend to verify how students approach programming learning, how they see its importance for their future, how much time they dedicate to programming, and which activities they use to develop their programming ability, among others. In these studies we mainly use three basic techniques, namely observations, interviews and questionnaires, as a way to collect facts and evaluate attitudes. That information will serve to improve the environment features. Anyway, there are a few important characteristics that we may already identify as crucial in an effective programming learning environment, namely:

- **Analysis and permanent update of each student knowledge level**. It is important that the environment always has updated information about each student knowledge level. The objective is to allow a more individualized and effective interaction with the student, both in the definition of meaningful activities proposed to her/him and also in the feedback given by the environment (for example using examples of problems already solved by the same student in the past).
- **Considering the student preferential learning style**. Another important factor, essential to make interaction more useful to each student, is the knowledge about each student learning style. This will influence the type of activities proposed and how they are presented to each student. There are different models to determine a persons predominant learning style, namely "The Myers-Briggs Type Indicator (MBTI)" [24], "The Kolb's Learning Style Model" [25] and "The Felder-Silverman Learning Style Model" [26], among others. We will use the "The Felder-Silverman Learning Style Model", since it is easy to apply through a computer-based questionnaire and it has its roots in the engineering field [27].
- **Use of programming patterns to help students**. A strategy to teach programming consists in using programming patterns. Thus, when the student shows incapacity to solve a particular problem or a part of it, the environment will present incomplete programs exemplifying some of the necessary components, instead of just waiting for students to write complete programs from scratch. The idea of using programming patterns is also to demonstrate some good practices. It is also an objective that students learn to transfer that knowledge to analogous situations.
- **Inclusion of games**. A crucial part of the environment, will consist in the inclusion of some types of ludic activities and logical games. The idea is to develop generic problem solving abilities, through attractive and stimulant activities, but also to attract students to the environment. Although currently the model is still not completely established, we were inspired by the non computational idea proposed in [28]. The methodology proposed by these authors includes basically three phases. The first works on the resolution of problems of different domains (symbolic puzzles, logical puzzles, games, brainteasers, enigmas, simple problems of arithmetic and geometry, among others) not working with algorithms or programming, but valuing the cognitive autonomy of students. Secondly, the

environment leads the student through the experience of formalization, valuing the concision and the precision of the used language. Finally, the objective goes to algorithm construction, intending to transform the developed formalization into systematic procedures. Gradually, the problems presented to the students start to demand more elaborate solutions, where the procedures will become more and more inherently detailed. Each of the phases will always be applied according to the current state of the student's knowledge and her/his preferential learning style.

- **Inclusion of an algorithm development environment**. When the student is in a more advanced phase of knowledge, where they are supposed to be able to create algorithms, the environment will present a tool for that propose. This tool will probably be an improved version of SICAS, a tool previously developed by the same team. SICAS is a system designed to support learning of the basic concepts of procedural programming. Its main objective is the development of problem solving skills, namely in the use of control structures and subprograms to solve problems. This support system includes two different activities:
  - Design/Edition of solutions (algorithms) to teacher proposed problems. The algorithm is specified using a visual representation, where graphical symbols that represent the algorithms building blocks (selections, repetitions, procedures…) are used. This representation is independent of any programming language that may be used in the course, allowing students to focus on the algorithm design and not on any specific language syntax. It is also useful to convey the idea that a well designed algorithm can be implemented in several programming languages without a significant effort.
  - Execution/Simulation of solutions. The student can verify how his/her algorithm works. SICAS simulates the algorithm and shows its results using animation. Students can analyze how the algorithm behaves in detail and at their own pace, identifying and correcting eventual errors. SICAS does not include theoretical contents, but it consists of an environment for experimentation and discovery, which enables the detection of errors, their correction and the learning based on these activities. In our opinion, these activities improve problem solving skills resulting in the ability to build programs. In several tests we confirmed that the students who used SICAS built better algorithms and made them faster than those who did not use the environment [29].

Once a valid solution has been created, SICAS allows the automatic generation of the corresponding code, using pseudo-code, C language or Java language. A more complete explanation can be found in [30].

Thus, in the last learning phase, the student would continue to solve problems but now expressing solutions in a programming-like way.

Besides supplying a personalized and permanent support to the student, the environment must have a set of alternative forms of representing a problem and its solution, adapting itself to the student learning style.

CONCLUSIONS AND FUTURE WORK

Low problem solving skills is a factor that, in our opinion, led a good number of students to failure and frustration in their introductory programming courses. In this paper we mentioned some educational tools proposed to support programming teaching and learning. Some of them have proved to be effective in particular topics. However, none of them attacks common problem solving difficulties efficiently to most students. So, we presented the reasoning beyond a new project we are developing. The objective is to better support programming learning, giving emphasis to problem solving activities according to students' learning styles. As the objectives are pedagogical, we also intend to do pedagogical evaluations related with study and teaching methodologies, in order to

improve, add or exclude some environment features. Once completed, the environment will be fully evaluated, especially in pedagogical terms.

REFERENCES

[1] Soloway, E. and J. Spohrer. (1989). Studying the Novice Programmer. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[2] Jenkins, T. (2002). On the difficulty of learning to program. In Proc. of the 3rd Annual LTSN_ICS Conference (Loughborough University, United Kingdom, August 27-29, 2002). The Higher Education Academy, Pp. 53-58.

[3] Lahtinen, E., Ala-Mutka, K. e Järvinen, H-M. (2005). A study of difficulties of novice programmers. In Proc of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica, Portugal, June 27-29, 2005, Pp. 14-18.

[4] Hanciles, B., Shankararaman, V. and Munoz, J. (1997). Multiple representation for understanding data structures. Computers & Education, Vol. 29 No. 1, Pp. 1-11.

[5] Brown, M. (1988). Exploring algorithms using BALSA-II. IEEE Computer, Vol. 21 No. 5, Pp. 14-36.

[6] Mendes, A. and Mendes, T. (1988). VIP - A tool to VIsualize Programming examples. In Proc. of the EACT 88 - Education and Application of Computer Technology, EACT 88, Malta, October 1988.

[7] Stasko, J. (1992). Animating algorithms with XTANGO. SIGACT News, Vol. 23 No. 2, Pp. 67-71.

[8] Levy, R. B., Ben-Ari, M., Uronen, P. A. (2003). The Jeliot 2000 program animation system. Computers & Education, Vol. 40 No.1, Pp. 1–15.

[9] Korhonen, A., Malmi, L., Silvasti, P. (2003). TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In Proc. of the 3rd Finnish/Baltic Sea Conference on Computer Science Education, Koli, Finlândia, Pp. 48-56.

[10] Cilliers, C., Calitz, A. and Greyling, J. (2005). The effect of integrating an iconic programming notation in CS1. In Proc of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica, Portugal, June 27-29, 2005, Pp. 89-93,

[11] Calloni, B. A. and Bagert, D. J. (1993). BACCII: An iconic syntax-directed system for teaching procedural programming. In Proc. of the 31st ACM Southeast Conference, Birmingham AL, April 15-16, 1993, Pp. 177-183.

[12] Kolling, M., Quig, B., Patterson, A. and Rosenberg, J. (2003). The BlueJ system and its pedagogy. Journal of Computing Science Education, Special Issue of Learning and Teaching Object Technology, Vol. 12, No. 4, Pp. 249–268.

[13] Ellis, G. P. and Lund, G. R. (1994). 2nd AL1-Ireland Conference on teaching of Computing, Dublin City University, Dublin, September 1994.

[14] Naps, T. (2005). Jhavé – Supporting Algorithm Visualization. IEEE Computer Graphics and Applications, Vol. 25 No. 5, Pp. 49-55.

[15] Anderson, J. R. e Reiser, B. J. (1985). The LISP tutor. Byte, Vol. 10 No.4, Pp. 159-175.

[16] Song, J. S., Hahn, S. H., Tak, K. Y. e Kim, J. H. (1997). An intelligent tutoring system for introductory C language course. Computers & Education, Vol. 28 No. 2.

[17] Moritz, S. H., Wei, F., Parvez, S. M., Blank, G. D. (2005) From objects-first to design-first with multimedia and intelligent tutoring. In Proc. of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica , Portugal, Vol. 37, No. 3, Pp.99-103.

[18] Papert, S. (1980). Mindstorms, children, computers and powerful ideias. New York: Basic Books.

[19] Pattis, R. (1981). Karel the Robot: A gentle introduction to the art of programming. (2nd Edition), John Wiley & Sons.

[20] Brusilovsky, P. (1994). Program visualization as a debugging tool for novices. In Proc. of INTERCHI'93, Amsterdam, 24-29 April 1993, Pp. 29-30.

[21] Jehng, J., Shih, Y., Liang, S. e Chan, T. (1994). Turtle-Graph: A computer Supported Cooperative learning environment. In Proc. of the ED-MEDIA'94, Pp. 293-298.

[22] Cooper, S., Dann, W., Pausch, R. (2003). Teaching objects-first in introductory computer science. In Proc. of the 34th Annual SIGCSE Technical Symposium on Computer Science Education, Reno, Navada, USA, 2003, Vol. 35, No. 1, Pp. 191-195.

[23] Gomes, A. (2002). Ambiente de suporte à aprendizagem de conceitos básicos de programação. Tese de Mestrado em Engenharia Informática, Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

[24] Myers, I. B. and McCaulley, M.H. (1985). Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator. Palo Alto, CA: Consulting Psychologists Press.

[25] Kolb, D. A. (1985). Learning Style Inventory: Technical Manual. McBer and Company, Boston.

[26] Felder, R. M. (1988). Learning and Teaching Styles in Engineering Education. Journal of Engineering Education, Vol. 78 No. 7, Pp. 674-681.

[27] Carmo, L., Gomes, A., Pereira, F. and Mendes, A.J. (2006). Learning styles and problem solving strategies. In Proceedings of 3rd E-Learning Conference – Computer Science Education (CD-ROM), Coimbra, Portugal, September 2006.

[28] Delgado, C., Xexeo, J. A., Souza, I. F., Campos, M. F. e Rapkiewicz, C. E. (2004). Uma Abordagem Pedagógica para a Iniciação ao Estudo de Algoritmos. Anais do Curso de Ciência da Computação, Vol. V, Pp. 72-87, Publicação semestral do Curso de Ciência da Computação das Faculdades Integradas Bennett.

[29] Rebelo, B., Marcelino, M. J., Mendes, A. J. (2005). Evaluation and utilization of SICAS – a system to support algorithm learning, In Proceedings of CATE05 – Computers and Advanced Technology in Education, Oranjestad, Aruba, August, 2005.

[30] Marcelino, M.J., Gomes, A., Dimitrov, N. and Mendes, A.J. (2004). Using a computer-based interactive system for the development of basic algorithmic and programming skills. In Proceedings of CompSysTech 2004 - International Conference on Computer Systems and Technologies, Rousse, Bulgaria, June 2004 Pp. IV.8.1-IV.8.6.

ABOUT THE AUTHORS

Adj. Prof. Anabela Gomes, MSc, Department of Informatics Engineering and Systems, Polytechnic Institute of Coimbra and CISUC – Department of Informatics Engineering, University of Coimbra, Phone: +351 239 790 350, E-mail: anabela@isec.pt.

Assist. Prof. António José Mendes, PhD, CISUC – Department of Informatics Engineering, University of Coimbra, Phone: +351 239 790000, E-mail: toze@dei.uc.pt.