Learning to program - difficulties and solutions

Gomes, Anabela[1,2], Mendes, A. J.[2]
[1]ISEC – Engineering Institute of Coimbra - Polytechnic Institute of Coimbra
anabela@isec.pt
[2]CISUC – Department of Informatics Engineering - University of Coimbra
toze@dei.uc.pt

Abstract **- Programming is a fundamental part of computer science curriculum, but it is often problematic. The high drop out and failure rates in introductory programming courses are a universal problem that motivated many researchers to propose methodologies and tools to help students. Although some of these tools have been reported to have a positive effect in students learning, the problem still remains mostly unsolved. We think that there are several reasons that cause this learning problem. Maybe the most important is the lack of problem solving abilities that many students show. They don't know how to program, because they don't know how to create algorithms, mainly due to their lack of general problem solving abilities. This and other causes to student difficulties are discussed in this paper. Some possible solutions are proposed, so that problems can be reduced.**

Index terms - Educational Technology, Learning Styles, Programming Teaching and Learning,

INTRODUCTION

It is well known that many students have difficulties in programming learning. Programming is a very complex subject that requires effort and a special approach in the way it is learned and taught. To become a good programmer, a student must acquire a series of abilities that go well beyond knowing the syntax of some programming language.

Several approaches and tools have been proposed aiming to support programming learning in different ways. Although we find reports of positive results as an outcome of some tools [1], none of them has a general use. In fact, the problem remains relatively unchanged as we continue to find reports about the difficulties many students experience when learning basic programming.

Experience shows that the problem starts for many students in the initial phase of learning, when they have to understand and apply abstract programming concepts, like control structures, to create algorithms that solve concrete problems. Particular attention is necessary in this initial stage, not only in the development of programming specific abilities, but also (and maybe above all) in the improvement and/or consolidation of knowledge and abilities that should have been acquired in previous years. These include generic problem solving abilities, logic reasoning and so on.

THE PROBLEM

I. The teaching methods

The traditional teaching methods do not seem adequate for many students needs, for different reasons:

- **Teaching is not personalized**. It would be desirable to have a teacher always available to allow more personalized student supervision. Immediate feedback during problem solving and detailed explanation of less understood aspects could probably help many students. However, in reality it is impossible to give this type of support due to time constraints and common course sizes.
- **Teachers' strategies don't support all students' learning styles**. People learn in several ways and have different preferences to approach new materials. In traditional education all students must learn at the same rhythm and in accordance to the teacher's pedagogical strategies, which are based on the teacher's learning style. Different students have different learning styles and can have several preferences in the way they learn. Some may regard learning as a solitary process while others may prefer a more dynamic learning environment, for instance through discussions with their peers. Additionally, some subjects may demand a particular learning approach but, without guidance, students will tend to adopt the style they prefer or which has served them best in the past. It is an important responsibility for the teacher to ensure that the students adopt the most appropriate learning approach for the subject at hand [2].
- **The teaching of dynamic concepts through static materials**. Programming involves several dynamic concepts that many times are taught through static means (projected presentations, verbal explanations, diagrams, blackboard drawings, texts, and so on). For some students this is a problem, as they fail to understand program dynamics through this type of materials.
- **Teachers are more concentrated on teaching a programming language and its syntactic details, instead of promoting problem solving using a programming language**. The purpose of an introductory programming course should be to increase students' programming abilities. However, many times teachers and students focus more in the programming language syntactic details. The language should only serve as a tool to express ideas and algorithms. However, an enormous amount of syntactic details are taught, normally before the students have a good

understanding of some important programming concepts. In our view, the language used in introductory programming courses should be chosen considering pedagogic suitability and not popularity in industry or some other reason.

II. The study methods

We consider that the study methods followed by many students are not suitable for programming learning. We can identify several aspects where improvement could happen:

- **Students use incorrect study methodologies**. Many students are used to solve problems from other disciplines through the memorization of formulas or procedures. Sometimes students memorize formulas, without a complete understanding of the underlying concepts, just knowing that a particular formula should be used to some type of problem. Programming requires a different study method. It should be essentially practical and very intensive, quite different from what is required in many other courses (more based in theoretical knowledge, implying reading and some memorization). Some students believe that they can learn to program mostly through reading a textbook, failing to understand that their main activity should be solving as many programming problems as possible.
- **Students don't work hard enough to acquire programming competences**. They are used to subjects where assisting classes and studying a text book is enough. However programming demands intense work extra classes.

III. The student's abilities and attitudes

- **Students don't know how to solve problems**. We think that the most important cause to the difficulties many freshmen feel to learn programming is their lack of generic problem solving skills. The students don't know how to create algorithms, mainly because they don't know how to solve problems. Problem solving requires multiple abilities that students often don't have, namely:
  i) Problem understanding - Many times the students try to solve a problem without completely understand it. Sometimes this happens because the student has difficulties interpreting the problem statement and others simply because students are too anxious to start writing code and don't read and interpret correctly the problem description.
  ii) Relating knowledge - Many students don't establish correct analogies with past problems and don't transfer prior knowledge to the new problems. They tend to group the problems that have the same superficial characteristics instead of the same principle. Consequently, many times students base their solutions on unrelated problems, leading to incorrect solutions.
  iii) Reflection about the problem and the solution - The students have a tendency to write an answer before thinking carefully about it. Many times testing is done superficially and they get satisfied just because the program works with a data set, without making more extensive testing.
  iv) Lack of persistence - Students often give up solving a problem if they don't quickly find a possible solution. Usually, solving programming problems demands effort and persistence. However, when facing any difficulty, many students prefer to ask the solution to a colleague or simply give up, instead of keep trying solving the problem. This is especially important, since learning is more effective when students find the solution, instead of simply reading the solution.
- **Many students don't have enough mathematical and logical knowledge**. Gomes et al. [3] conducted some experiments exploring the relationships between mathematical problem solving competences and the lack of programming abilities shown by a group of students that didn't get approved in their initial programming course. This experience was carried out during the second semester of 2005/2006 and the authors concluded that the involved students had deep difficulties in several areas, such as basic calculus and number theory or simple geometric and trigonometric concepts. The authors also report difficulties in transforming a textual problem into a mathematical formula that solves it. Limitations in abstraction level and logical reasoning were also identified.
  We think that mathematical knowledge is very important for programming and it is possible to find studies ([4], for example) that evidence some relationship between programming skills and experience in mathematics.
- **Students lack specific programming expertise.** Many students' programming difficulties are also caused by programming specific errors and misconceptions. Sometimes we find students that don't know how common programming structures work or have misconceptions about them. It is also common that students demonstrate difficulties to detect simple syntactical and logical programming errors.

IV. The nature of programming

- **Programming demands a high level of abstraction**. Programming learning requires skills like abstraction, generalization, transfer and critical thinking, among others. Experience has also shown that the problem starts, in general, in the initial learning phase, when students are expected to understand and apply certain abstract programming concepts, like control structures, to solve problems.
- **Programming languages have a very complex syntax**. Programming languages were developed for professional use and not to support learning. Common languages are extensive and have many complex syntactic details to memorize. That complexity requires that students have to concentrate simultaneously in algorithm construction and syntactic rules.

V. Psychological Effects

- **Students don't have motivation**. Many students don't have enough motivation to study programming, because there is an extremely negative connotation associated with programming that passes from student to student. There is the public image of a "programmer" as a

socially inadequate "nerd" [2]. Additionally, programming courses acquire the reputation of being difficult. So, it is hard to imagine students aspiring to this image. If students approach a course with an expectation that it will be difficult, and with a negative image of those who excel in the subject, it is very hard to imagine them as being especially motivated. And students who don't have an intrinsic motivation will hardly succeed. [5].

- **Students have to learn programming in a difficult period of their life**. Programming is normally taught as a basic subject in the beginning of a higher education course, coinciding with a period of transition and instability in the student's life. Some authors consider that programming disciplines are badly located in the curriculum, because this is a time of many difficulties and novelties to a new and autonomous life. The type of subject is already difficult enough when students are stable so, when placed in a period of transition this can only contribute to an increase in difficulty [2].

OUR PROPOSAL

How to solve or at least minimize each of the above discussed difficulties? In our view, this can be achieved through the development and utilization of a computational environment that may support students effectively. To reach this general objective it is necessary to define which characteristics are necessary to help in each of the mentioned problems.

- As **teaching is not personalized** the environment needs to provide permanent student supervision. The ideal situation would be to have a tutor to follow student's evolution, clarify doubts, and propose problems and activities. Another important role of the tutor is to prevent situations that may lead students to give up or lose motivation. A computational tutor may be beneficial in the sense that it won't show negative sentiments and will always show some tolerance! One of the teacher's roles is to motivate students. However, sometimes this is not easy and motivation lacks in both parts. Although a computational tutor cannot completely replace a human tutor, we believe it can contribute to captivate the students' attention, keeping them interested and allowing them to do the activities without inhibitions!
- **Teachers' strategies don't support all students' learning styles.** The environment must adapt activities to each student preferential learning style. This can be done having different presentation formats to each activity and adapting interaction strategies to the student characteristics. The first time students access the environment they will be asked to register and fill a questionnaire that allows the identification of that particular student learning style. There are different models for this purpose, for example "The Myers-Briggs Type Indicator (MBTI)" [6], "The Kolb's Learning Style Model" [7], and "The Felder-Silverman Learning Style Model" [8]. The later is widely used and easy to implement in a computational platform. As its origins are in the engineering field, we believe it is a good choice for our environment.
- **The teaching of dynamic concepts is usually made through static material.** The environment includes dynamic computational models representing programming concepts. However these models should be in accordance with the preferential learning style of the student.
- **Teachers are sometimes more concentrated on teaching a programming language and its syntactic details, instead of promoting problem solving using a programming language.** The environment includes multimedia tools focused on problem solving and algorithm development. Algorithms can then be implemented in any common programming language. Progress through the different types of problems must be gradual and progressive. In a first phase the problems are simple and have some playful dimension to attract and stimulate students. Gradually problems progress to more specific domains towards typical programming problems. Each new problem presented to the student demands more elaborated solutions, involving more detailed procedures. The main idea consists in promoting and evaluating the student's progress, through a stimulating and attractive system.
- **Students often use incorrect study methodologies** and **don't work hard enough to achieve success in this type of discipline.** To help in this area the environment must be attractive and include activities that may interest students. It has many practical problem solving activities, giving students the possibility to practice with a lot of diversified activities. Programming is "problem solving intensive" [9] requiring a significant amount of effort and different skills. So it is very important to give students opportunities to practice a lot. It is also important that after concluding an activity the environment provides additional questions or activities to make sure that the student understood the solution completely. That will allow students to reflect on their solutions and how they could be improved. This is an important activity that students rarely do. These consolidation/reflection questions can be of several types, for example including textual questions, graphical representations, and simulations involving some data changes.
- As **Students don't know how to solve problems** the environment includes many problems in accordance with the non computational model proposed in [10]. This environment must incorporate the following characteristics:

  i) For each problem, the environment must verify that the student understood what was asked. This can be verified, for example, asking the students what the input is and output data for the problem or asking them to predict the new output after some changes made to the problem.

  ii) The problems proposed have an increasing difficulty level maintaining, when possible, some connection with previous problems. At least the new problems should maintain some concepts needed to solve some subtasks

in previous problems. So, if the environment detects that the student isn't able to solve the current activity it can give hints, such as dividing the problem and connecting some of the parts with problems previously solved by the same student. Alternatively, it can also propose simpler challenges for further practice.

iii) When the student proposes a solution to a particular problem, she/he will have to face a set of questions or mini-activities in order to reflect on the proposed solution. These activities may include, for example, the adaptation of the student solution, so that it works also to some variation of the original problem.

iv) Integrating mechanisms that prevent the students from giving up solving problems. This objective is pursued through a lot of interactivity with the student, generating adequate feedback, whenever it is necessary. The idea is to motivate and encourage them to continue trying to solve the problem. This can be done in different ways. For instance, after a long period of student's inactivity, the environment can present the complete solution, asking questions about it. Alternatively, students can be asked to discover some errors on a presented solution or to complete parts of an incomplete solution. The system can also suggest smaller challenges, based on the problem, making it easier to solve.

- **Many students don't have enough mathematical and logical knowledge.** The environment includes a set of challenges that include implicit or explicit mathematical concepts, especially those concepts that are important for typical programming problems.
- **Students lack specific programming expertise.** Many times there is a gap between generic problem solving and programming problem solving. Hence it is necessary that the environment helps students to make this transition. This can be done essentially in two ways, namely:

  i) Helping and giving hints to students, teaching them certain aspects of programming problem solving. To do this the environment can present complete programs for students to analyze. We think that the best way to learn to program is trying to write programs from scratch. However, to study and to test complete programs can help students to understand how programs work. Also analyzing the strategies used to solve some problems can help students in their initial phase of learning to program. Another useful activity is to analyze programs that include logical errors usually made by students. Completing incomplete programs can also be a useful activity in initial stages, instead of waiting for students to write entire programs from the beginning. The environment also includes programming patterns, representing solutions of common problems in a format that exemplifies good programming practices.

  ii) Allowing students to test their principles, theories and reasoning. The environment allows the simulation of students' algorithms, so that they can verify program behavior and logical errors. This is done through the inclusion of the SICAS environment [11], which allows animated simulation of student built algorithms.

  The environment is based on a constructivist approach of learning, where each student learns at her/his own pace and progressively constructs her/his own knowledge. We strongly believe, like many researchers, that this kind of approach can improve student problem solving abilities, as well as their critical thinking capabilities [12].
- **Programming demands a high level of abstraction.** It is important that the environment helps to develop the student's abstraction capacity. For that, it includes from everyday problems to more specific problems concerning the programming domain. It is important that students learn to recognize patterns in the different problems, developing their generalization skills. We think that the non computational model proposed in [10] promotes the gradual development of students' abstraction capacity, helping them to relate new situations with their previous experiences.
- **Programming languages have very complex syntaxes.** The environment minimizes aspects inherent to language syntaxes, emphasizing the algorithmic and problem solving processes. In this way it creates conditions for students to concentrate essentially in solutions without having to take care of complex syntaxes.
- **Students don't have motivation.** We propose a multimedia environment that includes several types of problem solving activities. In the initial stages the activities will have a more playful nature, using knowledge from diverse domains, as a way to attract students to the environment. When the student shows some domain of basic problems, the environment will propose problems that demand more complex solutions, including typical programming problems. The objective is not proposing problems that in a given stage are too difficult for the student, eventually causing her/him to lose motivation. It is also important to show the students that programming is a useful tool to ease people life. This can be achieved using real life problems as much as possible.
- **Students have to learn programming in a difficult period of their life**. In our opinion, programming should be preceded by an intensive training in problem solving. Hence, we think that students should follow at least a course devoted to problem solving before they engage in typical programming courses. However, the Bologna process lead to 3 years programs in many institutions. As programming is a pre-requisite to many other courses, it is necessary that it appears early in the curriculum. In this context, we think that an environment with a strong problem solving emphasis can help reduce this problem.

The environment described in this paper is in an initial development phase. That is why we presented essentially its specification and the reasoning behind it. As the environment objectives are mainly pedagogical, we are conducting a few experiments to help us better define its characteristics and to more precisely identify the causes of student's difficulties.

We had already conducted some experiments [11, 14], trying to determine how the development of mathematical

and logical problem solving abilities would impact programming capacity. In this study we could identify many students' problem solving difficulties and relate them with their programming limitations. These results were analyzed taking into consideration the students learning styles preferences. So we already have a precise idea of the type of activities that should be incorporate and the way to do so according to the different learning styles.

However our experiments were restricted to problem solving aspects and we did not address the way students acquire knowledge. 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 students' problem solving capacity. We also plan to investigate how to improve teaching strategies in introductory programming courses, taking into account students' learning styles.

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 learning, 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.

Once completed, the environment will also be fully evaluated, especially in pedagogical terms. It will be necessary to see how different students use the different materials and activities and to evaluate their impact in the development of the student's programming abilities.

CONCLUSION

There are different reasons why programming learning is inherently difficult. The question is somewhat complex. We agree with some authors that say that programming requires not a single, but a set of skills. Those skills form a hierarchy [13] and a programmer will be using many of them at any point in time. In our opinion, the most important for novice programming students is to develop their problem solving abilities. So we are developing a computational environment mainly based on problem solving activities from different domains. When the student reaches a higher competence level in generic problem solving, the environment starts to propose typical programming problems. We believe this is the best approach as programming education should be preceded by the development of a sound problem solving competence.

The environment also tries to adapt itself to each particular student characteristics, namely taking into consideration her/his previous work and preferred learning style when selecting activities and interaction modes that will be used with that particular student.

REFERENCES

[1] A. Lawrence, A. Badre and J. Stasko, "Empirically Evaluating the Use of Animations to Teach Algorithms", in *Proc. of the IEEE Symposium on Visual Languages*, St. Louis, MO, October 1994, pp. 48-54.

[2] T. Jenkins, "On the difficulty of learning to program", in *Proc. of the 3rd Annu. LTSN_ICS Conf.*, Loughborough University, United Kingdom, August 2002, pp. 53-58.

[3] A. Gomes, L. Carmo, E. Bigotte and A. J. Mendes, "Mathematics and programming problem solving", in *Proc. of the 3rd E-Learning Conf. – Computer Science Education (CD-ROM)*, Coimbra, Portugal, September 2006.

[4] P. Byrne and G. Lyons, "The Effect of Student Attributes on Success in Programming", in *Proc. of 6th Annu. Conf. on Innovation and Technology in Computer Science Education - ITiCSE 2001*, United Kingdom, 2001, pp. 49-52.

[5] C. Bereiter and E. Ng. "Three Levels of Goal Orientation in Learning", *Journal of the Learning Sciences*, vol. 1, nº 3, pp. 243-271, 1991.

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

[7] D. A. Kolb, *Learning Style Inventory: Technical Manual*. McBer and Company, Boston, 1985.

[8] R. M. Felder and L. K. Silverman, "Learning and Teaching Styles in Engineering Education", *Journal of Engineering Education*, vol. 78, nº 7, pp. 674-681, 1988.

[9] D. N. Perkins, S. Schwartz and R. Simmons, "Instructional Strategies for the Problems of Novice Programmers", in R. E. Mayer (ed.), *Teaching and Learning Computer Programming*, Lawrence Erlbaum Associates, 1988, pp. 153-178.

[10] C. Delgado, J. A. Xexeo, I. F. Souza, M. F. Campos and C. E. Rapkiewicz, "Uma Abordagem Pedagógica para a Iniciação ao Estudo de Algoritmos", *Anais do Curso de Ciência da Computação*, vol. V, pp. 72-87, 2004.

[11] A. Gomes and A. J. Mendes, "SICAS: Interactive system for algorithm development and simulation", in Manuel Ortega and José Bravo (ed.), *Computers and Education in an Interconnected Society*, Kluwer Academic Publishers, 2001, pp. 159-166.

[12] M. Ben-Ari, "Constructivism in Computer Science Education", *Journal of Computers in Mathematics & Science Teaching*, vol. 20, nº 1, pp. 45-73, 2000.

[13] K. D. Sloane and M. C. Linn, "Instructional Conditions in Pascal Programming Classes", in R. E. Mayer (ed.), *Teaching and learning computer programming*, Lawrence Erlbaum Associates, 1988, pp. 137-152.

[14] L. Carmo, A. Gomes, F. Pereira and A. J. Mendes, "Learning styles and problem solving strategies", in *Proc. of the 3rd E-Learning Conf. – Computer Science Education (CD-ROM)*, Coimbra, Portugal, September 2006.