Problem solving in programming

Anabela Gomes[1,2]
António José Mendes[2]

[1]Department of Informatics Engineering and Systems, Polytechnic Institute of Coimbra
anabela@isec.pt

[2]CISUC – Department of Informatics Engineering - University of Coimbra
toze@dei.uc.pt

Abstract. We think that the major cause of the students' failure in introductory programming course is the lack of a basic skill, the problem solving ability. Several authors frequently regarded this skill as the most important cognitive activity in everyday, professional and educational contexts. In traditional programming teaching, generic problem solving is not emphasized. In this paper we discuss the concepts and stages of problem solving, considering also how experts and novices solve problems. The idea is that this analysis leads to a number of important principles for teach and learn problem solving strategies. The main purpose of this paper is to present the features of a system currently under development to support programming learning, focusing in problem solving activities.

1 Introduction

Learning to program is a difficult process to many students. This causes a high failure rate in many institutions worldwide. Several authors have discussed different reasons for such problems [1, 2, 3, 4, 5]. In general they argue that programming is difficult due to the nature of the subject, as programming requires a hierarchy of skills like abstraction, generalization, transfer and critical thinking, among others. Also, some authors refer that teachers' methodologies often doesn't take into consideration the dynamic nature of programs, as they are presented mostly using materials. Several authors also refer that many students don't follow an adequate study method for programming learning. Programming requires a very practical and intensive approach, which is quite different from what is required in many other courses (more based in theoretical knowledge, implying extensive reading and some memorization). Student background is often mentioned as a cause of problems, especially in what concerns problem solving skills. Often authors point out that the main difficulty for many students is to create solutions to problems, that is creating algorithms to solve them. In other words, the main difficulty is to devise and formalize a solution and not its codification in a particular programming language. We believe that these problem solving difficulties are not specific of typical programming problems, but more general to other types of problems. Therefore, in this paper we emphasize problem solving processes. We will discuss the stages and strategies involved in problem solving, the necessary abilities, as well as different approaches for problem solving. The difference between novice and expert students when they are involved in problem solving tasks in different domains will also be mentioned. Perhaps, this comparison may help to answer some questions, such as, *Is it possible to teach how to solve programming problems? What abilities a novice needs to acquire to become an expert in programming?*

2 Problems and Problem solving

In our view, to improve programming teaching and learning it is fundamental to develop student's problem solving abilities. Based on this assumption we did a literature review about several aspects connected with problems and their solutions.

2.1 What is a problem?

For a better understanding of the processes involved in problem solving it is useful to know how different authors define what a problem is. For Gagné [6], it is a process where the apprentice/learner discovers a combination of rules previously learned that he/she can apply to reach a solution for a new problematic situation. Hayes [7] defined a problem as the gap that separates a present state from a desired state. Gil Pérez et al. [8] consider a problem as a situation for which there isn't an evident solution. Perales [9] considers a problem as any situation that produces, on one hand, a certain degree of uncertainty and, on the other, behaviour in search of a solution. For Mayer [10], it is a multifaceted and complex concept, with cognitive, metacognitive and motivational aspects. The majority of the researchers in this area consider problem solving as a subjective state of mind. It is different from individual to individual, constituting a challenge, a situation not solved, whose reply is not immediate. Reflection, use of conceptual and procedural strategies is needed in order to provoke a change in the learner mental structures that may lead to problem solution. Finally, Flavell [11] says that it is not worth to define "problem" as it is something complex and instable, requiring creativity to its solution.

2.2 Stages in Problem Solving

To verify where students fail during programming problem solving it is useful to divide the problem solving process in stages. A literature review on this subject allows us to find theories from different ages and knowledge domains. For instance, Descartes [12] in his "Discourse on Method" proposed a new "method" of thinking or problem solving. His method consisted of four rules, which Descartes writes in the informal, first-person style of the entire Discourse: i) "to accept nothing as true that I did not know to be evidently so"; ii) "to divide each of the difficulties I was examining into as many particles as I could"; iii) "to conduct my thoughts in order, beginning with the simplest objects and those that are easiest to understand, and progressing, as if by degrees, to the understanding of the most compound" and iv) "always to carry out enumerations so complete and reviews so general, that I would be certain of having omitted nothing."

Polya, in his book "How to Solve It" [13], describes many ideas on how to help students learn. The main ideas in the book are about problem solving which Polya thinks of as a four-phase process, namely: i) understand the problem; ii) devise a plan - it often means looking at related or simpler problems; iii) carry out the plan and iv) look back.

Hyman and Anderson [14] say that the essential for a successful problem solving is, first of all, to run over all the elements of the problem in rapid succession, many times, until a pattern emerges and don't jump hastily to conclusions (similar with Descartes' first rule).

Also in the scope of information processing theories some models including stages of problem solving processes have been proposed. Although the steps are different, on the whole, they express a similar sequence. According to Sternberg and Davidson [15] these steps are: i) problem identification; ii) selection of the mental operation to solve it with success; iii) internal and external representation of the information, in a clear way; iv) selection of an adequate strategy; v) distribution of the available resources; vi) monitor the different moments of problem solving, this is, having the conscience of what we made, what we are making and the evaluation of the solutions. In fact, problem solving methods are often described in a logical sequence of stages [16] that, together, describe the phases postulated by information processing models: i) input, where the problem is perceived and an attempt for understanding the situation or the problem is made; ii) processing, the phase where alternatives are generated and evaluated and a solution is selected; iii) output includes the planning and the implementation of the solution; iv) revision, where the solution is evaluated and the necessary modifications are made.

Pretz and colleagues [17] also divided the problem solving process in several stages: i) to recognize or to identify the problem; ii) to define and to represent the problem mentally; iii) to develop a resolution strategy; iv) to organize the knowledge concerning the problem; v) to attribute mental and physical resources to solve the problem; vi) to monitor ideas so not to divert from the main goal; vii) to evaluate and correct the solution.

Bransford and Stein [18] proposed the IDEAL model, which includes the following stages: i) **I**dentification of the problem; ii) **D**efinition of the problem with precision; iii) **E**xploration of strategies to reach the problem solution (based in previous knowledge and experiences); iv) **A**ction, in the sense of the execution of the previously planned; v) **L**earn (or Looking back) relative to the observation of the effect of the carried through actions and learning according to the evaluation of the results of these actions.

Also Santucci [19] synthesized in the acronym FARE several problem solving techniques and methods that basically retake the original model of Polya, referring the following stages: i) **F**ocusing on the creation, selection and definition of the problem, deciding what is necessary to know; ii) **A**nalysing, by collecting reference data, determining the relevant factors, and generating alternative solutions (or action plan); iii) **R**esolving, by selecting one solution, developing a plan for update and persistence in the organization to reach the awaited result; iv) **E**xecution, finding a solution, controlling the impact during the plan implementation (evaluation of the results).

Almeida [20] proposes a prescriptive model of the problem solving process. It includes the following phases: i) Recognition, definition and identification of the problem; ii) Analysis of the problem and generation of alternative solutions; iii) Development of plans; evaluation of the alternatives and selection of one of them; iv) Selection and effective implementation of the alternative solutions; v) Evaluation and follow-up or solution testing.

Taking into consideration the latter models and also the OECD-PISA report [21][1], we can describe problem solving stages, and identify some obstacles that, according to our experience, students frequently face in each of them, when attempting to solve programming problems.

– *Understanding the problem*. This stage is fundamental for good problem solving, but usually many students neglect it, not dedicating enough time to it. Frequently students are already attempting to solve a programming problem when they realize that they really didn't understand the actual problem. In programming, where the problems are often badly-defined or incompletely described, this phase necessarily includes the problem definition and understanding all ambiguous or incompletely detailed aspects. This includes understanding text, diagrams, formulas or tabular information, drawing inferences from them and relating information from various sources. It is important to show understanding of the relevant concepts; and using information from students' background knowledge to understand the given information.

– *Characterising the problem.* This includes the way students identify the problem variables and their interrelationships, making decisions about which variables are relevant and irrelevant, constructing hypotheses, and retrieving, organising, considering and critically evaluating contextual information. This phase implies looking for related or analogous problems that students may have already solved or that are of their knowledge. This type of strategy is rarely or badly used by many students. They often don't use interrelated knowledge, and other times they misidentify related problems, as they use seemingly similar problems, but that imply very different resolution strategies.

– *Representing the problem*. This includes the way students construct tabular, graphical, symbolic or verbal representations of the problem. Using external representations and shifting between representational formats may help students to better develop quality solutions. The students should be more encouraged to represent the problems, using their favourite representations and those that better translates their understanding of the question.

– *Solving the problem*. This includes making decisions, designing a system to meet certain goals, diagnosing and proposing a solution. In order to solve a problem it is necessary that all parts decided previously are now linked into a coherent and correct whole. Independently of the different approaches (top-down, bottom-up...) used to solve problems it is now important to make an effort to relate them and to solve the

[1] The Organisation for Economic Co-operation and Development's (OECD) Programme for International Student Assessment (PISA) is an internationally standardised assessment that was jointly developed by several participating countries and administered to students in some schools of those countries. Learning for Tomorrow's World – First Results from PISA 2003 (OECD, 2004a) summarises results from the assessment of the problem-solving skills.

task completely. It is quite frequent that students tend to give up on the slightest difficulty, putting aside a problem just because they don't know how to solve one of its parts.

– *Reflecting on the solution*. This includes examining solutions and looking for additional information or clarification; evaluating solutions from different perspectives in an attempt to restructure them, so that they can be more socially or technically acceptable. At this stage it is also important to be able to justify produced solutions. Usually the student's main objective is to reach the fastest way to obtain a solution, not analyzing it later with care. This stage must include a compete analysis of the solution implying, sometimes, the additional search or clarification of some information. The evaluation of the solution from different perspectives must also be considered, in an attempt to reorganize the solution and optimizing it. We consider that programming is learnt not only by programming, but also through reflexion about the way we programmed the solution. Would we do it in a different way the next time? How did other people solve the same problem? In programming subjects, we also consider very useful the discussion of different student's solutions, in order to verify different points of view.

– *Communicating the problem solution*. This includes selecting appropriate media and representation to express and to communicate solutions to an outside audience. We think that trying to communicate the solution can help students to detect problems that were previously not understood and also to reflect about the produced solutions.

We believe that the above steps, if practised during programming learning, will lead to a more organized and disciplined approach that will help students in the subject.

The study published by Almeida [20] seems to confirm some of our ideas about student's difficulties. Although this study target was mathematics secondary education students, many conclusions are similar to our experience with programming students. The author concluded that, when asked to solve a problem, in only 50% of the cases the students completely defined the problem (they identified the variables and the unknown quantities or the problem initial and final states). In 32% of the cases they only defined the problem partially. According to solution planning, only in 47% of the cases the problem was properly planned. The author refers that many students showed a high difficulty to produce some helpful graphical representation of the problem. Most students only tried to create it after a suggestion from the teacher. The attempt to find the problem solution occurs usually immediately after a short contact with its description. In general, it was observed in this stage that the students tried to find a solution immediately, even before a careful analysis and understanding of what was required. This is a tendency we often noticed also in our programming students. In the study, the observing teacher often recommended students to pay attention and to meditate about their answers, but 9% of the answers appeared by insight, and these weren't always correct. It was also verified that only in a few cases the students showed a reflexive attitude about the problem or the solutions they propose. For example, in 45% of cases the students accepted the solution they produced, just because they reached a result, without worrying about its correction or coherence. The author also concluded that the students that better solve problems are those that better understand the statements and more frequently verify, in a controlled way, the process and progress of the resolution, or the coherence of the solution with the problem data and conditions.

3 Types of competences necessary to solve problems

To help students to become better problem solvers, it is important to understand which abilities are essential to solve problems. To define a taxonomic organization of the necessary cognitive abilities it is useful to consider the mind as a hierarchic and multidimensional universe, whose different levels of organization obey to different rules [22]. In this direction, problem solving is considered the higher ability of a complexity continuous [23, 24].

According to the OECD report [21], problem solving is a combination of some different cognitive processes, organized in order to achieve a certain goal that could not be reached, at least in an evident way, through the simple application of a procedure, a process, a routine or an algorithm, already known, from a unique area. Problem solving involves different abilities of reasoning, thus the process of problem solving not only involves the student's knowledge, but also their reasoning abilities.

For instance, when trying to understand a problem situation, the student may need to distinguish between facts and opinions. When formulating a solution, it is important to identify relationships among variables. When selecting a strategy, the student has to consider cause and effect. When solving a problem and communicating the result, the student has to organize information in a logical manner. These activities often require analytical reasoning, quantitative reasoning, analogical reasoning and combinatory reasoning skills.

The same report mentions that different reasoning implies diverse activities in different situations. Thus, analytical reasoning is necessary in situations where the learner has to apply principles from formal logic when he/she determines the problem necessary and sufficient conditions. Quantitative reasoning occurs when the student has to apply properties and procedures related with the perception of numbers and numerical operations. Analogical reasoning happens when the learner has to solve a problem that has a similar context to some other previously solved by the student. The parameters or the context of the new problem may be different, but the strategy and reasoning to apply are the same. The student should be able to solve the new problem, interpreting it according that past experience. Combinatory reasoning takes place when the learner has to examine some factors, to consider all combinations that can happen, to evaluate each one of them in relation to some objective limitations that may exist, and later to select or order the combinations hierarchically.

Kizlik [25] identified a set of abilities that he considers nuclear for an efficient cognitive performance, namely: focalization, collecting information, memory, organization, analysis, execution, integration and evaluation. The author also argues that it is important that students are able to integrate the different abilities of the various types. The more they can do it the better they solve problems. The less developed abilities, in turn, indicate aspects to take into consideration in any problem solving training program.

We agree with Sloane and Linn [26], when they state that programming doesn't consist in a unique ability, not even in a set of abilities, but the necessary abilities form a hierarchy and the programmer has to use many of them simultaneously. Which abilities are necessary? What types of abilities have the experts when solving problems that the novices do not have? In [27] it is possible to find an analysis that tries to find differences in the way experts and novices solve problems in different knowledge domains, namely chemistry, physics and mathematics. The conclusions are described next, also using similar conclusions presented by other authors.

- Some studies found that 'experts' use 'working-forward' or 'knowledge development' strategies while 'novices' use 'working-backward' strategies or 'means-ends analysis' [28, 29, 30]. 'Working forward' strategies imply that the solver operates from the given problem (initial state) to the goal (the desired answer) while 'working backward' strategies operate from the goal to the initial state [31]. So, experts working in this way construct a more complete representation of the problem because they have extra knowledge available.
- Reviewing the research in the 1980s, Mestre [32] concluded that experts have extensive knowledge that is highly organised and used efficiently in problem-solving. Experts also approach problem-solving differently from novices. They categorise problems qualitatively and according to major principles whereas novices categorise problems quantitatively and according to their superficial attributes (i.e., the objects that appear in the problem statement).
- Novices try to solve the problem immediately, spending little time in its interpretation, and sometimes they solve it by trial and error, because they do not possess much auxiliary knowledge. According to Neves and Anderson [33], practice can help shorten the time necessary to solve problems. It is more likely that experts can solve problems in less time, as they have developed automatic processing through a lot of practice [34]. Larkin et al. [29] claimed that experts combined principles, collected necessary information and generated new information all in a single step.
- Experts come close to the solution through a process of successive refinements, starting with a rude description of the problem, through words and drawings and only later they examine the details of the problem. The novices start with a search of some principle or equation, since they do not have interrelated knowledge as the experts have. Experts solve problems from a general principle, using the deductive reasoning.
- Larkin et al. [29] also verified that opposed to experts, novices analyze the problem superficially, through memorization tasks and they do not review their answers.
- While experts tend to perceive a problem as an analysis and reasoning task, novices try to find the answer or solution quickly.
- Novices tend to use rules incorrectly, to see clues where they do not exist, and to consider as valid the first option that appears. They often do not test hypotheses, many times use logic badly, ignore previous reasoning and don't reflect on the problem solution.

So, in order to improve problem-solving among students, we think that it is important to understand the advantages that expert problem-solvers have and transform these advantages into problem-solving directions. Perhaps by teaching expert's problem-solving procedures to novice students, they will be able to improve their abilities, approaching the knowledge framework of experts.

4 Our proposal

Over the years 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. 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 [35], 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. Many other systems have been proposed, using visual representations or algorithm animations. BALSAII [36], VIP [37], XTango [38], Jeliot 2000 [39], Trackla [40], BlueJ [41] and Jhavé [42] are known examples.

Artificial intelligence techniques to support programming teaching and learning have also been proposed, namely Intelligent Tutoring Systems, such as Lisp-Tutor [43] or C-Tutor [44].

Microworlds are another popular proposal, seriously influenced by the turtle graphics of LOGO [45]. Examples include "Karel the Robot" [46], Tortoise [47], TurtleGraph [48] and Alice [49].

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. Maybe many tools 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) [50]. 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.

Some experiments were made with SICAS [51]. In those experiments we verified that SICAS is useful for students with average skills, but not for weaker students, who have many difficulties related with mathematical and logical concepts and severe limitations concerning problem solving.

Based on the above considerations, we believe it is possible to reduce student's problem solving difficulties, through the utilization of a computer-based environment that proposes activities to the students according to their current level and preferred learning style. During problem solving the environment follows student's work, giving advice when necessary. This environment is currently under development. Its main objectives and strategies are described in this section.

This multimedia environment will integrate several types of problem solving activities. For beginning students the activities will have a more ludic nature, using knowledge from diverse domains, as a way to attract and to stimulate students. As the student progresses, the environment starts to propose problems that demand more elaborated solutions. Step-by-step, the activities will progress gradually towards programming problems. The main idea consists in continually verifying the student's progress, through a stimulating and attractive system. Student's progress must be analysed in terms of abstraction capacity, logical reasoning, problem solving capacity and cognitive autonomy. The final goal for the students is the construction of algorithms that solve typical programming problems, eventually using previously developed formalizations and transforming them into common programming representations, using a suitable environment, such as SICAS.

SICAS includes support to two types of activities: design/edition of solutions (algorithms) to proposed problems and execution/simulation of those solutions. In the first case, the student can construct algorithms using visual representations (flowcharts) where they use graphical symbols that represent the structures (selections, repetitions, functions…) necessary to build the algorithm. 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. In the second case, the student can simulate and animate the execution of the algorithms she/he designed, analysing with detail and desired rhythm the various phases and entities it involves. SICAS does not include theoretical contents, but it consists of an environment to support experimentation and discovery, which facilitates 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.

The activities proposed in each stage to a particular student should take into consideration her/his knowledge level and preferential learning style. The first time a student accesses the environment he/she will be asked to answer a questionnaire, allowing the system to determine that particular student preferred learning style. There are different models for this purpose, for example "The Myers-Briggs Type Indicator (MBTI)",[52], "The Kolb's Learning Style Model" [53], and "The Felder-Silverman Learning Style Model" [54]. We use "The Felder-Silverman Learning Style Model", mostly because its origins are in the engineering field and also because the diagnostic is based in a simple to apply questionnaire.

The environment emphasises the aspects previously referred as constituting the main obstacles for efficient problem solving. Firstly, it is important to make sure the student completely understands the proposed problem or activity, preventing any further development before that is accomplished. This is done, for example, asking the student to determine which facts are known and which are uncertain and/or what the input and output data are. If the student can't answer these questions the system summarizes the problem through different representations and according to the student learning style. The system must also determine if the student has the necessary background information to solve the problem. For instance, sometimes the resolution of a programming problem presumes knowledge about some mathematical concepts. If the student shows difficulties with those concepts, the environment uses challenges or activities that include them implicitly or explicitly. The system contains also tutorials, animations and different types of explanations about the corresponding topic. However, explanations will only be presented in real contexts. So the idea is to use authentic problems with scenario-based simulations and games and not to explain concepts in an independent, abstract and non contextualized way.

Sometimes it is difficult for a student to begin a resolution. In this case, the system must give suggestions to students according to their previously recorded knowledge. So, if necessary, the environment can list several approaches, so that students choose which ones to use in the particular problem. Initially, most presented methods are wrong to facilitate the student choice. As the student progresses several methods can be correct, but the student can be asked about the most suitable or the most efficient. As the student is becoming more confident the environment will remove this type of help. During a problem solving activity the system will also put questions, trying to lead students to divide the problem in smaller components. During the student activity the environment monitors her/his actions and tries to detect situations where the student is diverging significantly from the correct problem solution.

After the student completes a solution to the proposed activity, the environment puts a set of questions or mini-activities (in accordance with his/her learning style), trying to stimulate reflection on the proposed solution. The idea is to review the solution steps in order to revise and refine them. In some cases the environment can make suggestions, asking for small changes in the problem solution, so that it meets a wider set of situations. Finally the environment presents the best solution for the proposed activity, allowing the student to compare his/her own solution with it, identifying possible improvements that can be made.

Many times there is a gap between generic problem solving and programming problem solving. Hence it is necessary that the environment helps the students to make this transition. This is done through activities such as analysing complete programs, completing unfinished programs, detecting errors in programs and showing/using programming patterns. We think that the best way to learn to program is to program as much as possible, looking for solutions to typical problems. However, to study and test previously made programs (and the strategies used during development) can help students to understand how they work and give them experience to face similar problems. To finish incomplete programs and detect errors can also be a good help for students in some stages of their development.

We think that the use of patterns can also be very useful, especially when the student does not have any real programming experience. They can serve as good programming examples, but also as building blocks that may facilitate the development of new programs.

During problem resolution the environment reminds the students about some general principles or strategies which the student surely knows or that he/she used before in the solution of some other problem. The recognition of patterns or relationships in large amounts of information, the use of analogies and metaphors to explain a problem, are also used intensively.

5 Conclusion

Low problem solving skills is one of the factors that led a good number of students to failure in their introductory programming courses. Based on a literature review we concluded problem solving skills requires many abilities, like abstraction, generalization, transfer and critical thinking, among others. So, in this paper we described the main characteristics of a new computational environment that pretends to help programming learning through different problem solving activities. We specified each of the system features based on the student's difficulties mentioned by several authors, and on our experience as teachers. We hope that knowing these difficulties and taking them into consideration will make possible to help learning and teaching in an effective way, more adapted to each student needs.

References

1. Sloane, K. D. and Linn, M. C.: Instructional Conditions in Pascal Programming Classes. R. E. Mayer (Ed.), Teaching and learning computer programming. Hillsdale, New Jersey: Lawrence Erlbaum Associates, (1988) 137-152.
2. Gomes, A. e Mendes, A. J.: Ambiente de suporte à aprendizagem de conceitos básicos de programação. In Actas do 3º Simpósio de Investigação e Desenvolvimento de Software Educativo, Évora, Portugal, Setembro-1998.
3. Soloway, E. and J. Spohrer.: Studying the Novice Programmer. Lawrence Erlbaum Associates, Hillsdale, New Jersey (1989).
4. Jenkins, T.: 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, (2002) 53-58.
5. Lahtinen, E., Ala-Mutka, K. e Järvinen, H-M.: A study of difficulties of novice programmers. In Proc of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Pp. 14-18, Monte de Caparica, Portugal, June 27-29, (2005).
6. Gagné, R.M.: The conditions of learning. New York: Holt, Rinehart and Winston, (1965).
7. Hayes, J. R.: The complete problem solver. Philadelphia: Franklin Institute Press, (1981).
8. Gil Pérez, D., Martinez Torregrosa, J. e Senent Pérez, F.: El fracaso en la resolución de problemas de física: una investigación orientada por nuevos supuestos. Enseñanza de las Ciencias, Barcelona, Vol. 6 No. 2, (1988) 131-146.
9. Perales, F. J.: La resolución de problemas: una revisión estructurada. Enseñanza de las Ciencias, Vol. 11 No. 2, (1993) 170-178.
10. Mayer, R. E.: Cognitive, metacognitive and motivational aspects of problem solving. Instructional Science, Vol. 26, No. 1-2, (1998) 49-63.
11. Flavell, J. H.: Metacognitive aspects of problem solving. The nature of intelligence. L. B. Resnick (Ed.) Hillsdale, New Jersey: Lawrence Erlbaum Associates, (1976) 231-235.
12. Descartes, René.: Discours de la methode and Meditationes de prima philosophia, (1637), as quoted in Discourse on Method; and Meditations on First Philosophy, transl. D. A. Cross, Indianapolis, Hackett Pub. Co. (1993).
13. Cross, Indianapolis, Hackett Pub. Co. 1993. Polya, G.: How to Solve It. Princeton University Press, Princeton, New Jersey, (1945).
14. Hyman, R. and B. Anderson.: Solving Problems. International Science and Technology, (Sept. 1965) 36-41.

15. Sternberg, R. J. and Davidson, J.E.: A four-prong model for intellectual skills development. Journal of Research and Development in Education, Vol. 22, No. 3, (1989) 22-28.
16. Osche, R.: Before the gates of excellence, the determinants of creative genius. Cambridge, New York: Cambridge University Press, (1990).
17. Pretz, J. E., Naples, A. J. and Sternberg, R. J.: Recognizing, defining and representing problems. J. Davidson and R. Sternberg. (Ed.). The psychology of problem solving. New York: Cambridge University Press.
18. Bransford, J. D. and Stein, B. S.: The IDEAL problem solver: A guide for improving thinking, learning and criativity. New York: W. H. Freeman and Company, (1984).
19. Santucci, U.: Problem setting. From http://web.tiscaline.it/problemsetting/
20. Almeida, A. C.: Cognição como Resolução de Problemas: Novos horizontes para a investigação e intervenção em Psicologia e Educação. PhD Thesis. Faculdade de Psicologia e Ciências da Educação da Universidade de Coimbra, (2004) (in Portuguese).
21. OECD (Organisation for Economic Co-operation and Development). Learning for tomorrow's world. First results from PISA 2003, Paris, available in http://www.pisa.oecd.org/dataoecd/38/30/33707234.pdf
22. Demetriou, A. Nooplasis.: 0 + 1 postulates about the formation of mind. The Journal of the European Association for Research in Learning and Instruction [Special issue: Learning and Instruction], Vol. 8, No. 4, (1998) 271-287.
23. Gagné, E.: The cognitive psychology of school learning. Boston: Little Brown and Company, Boston, (1985).
24. Seamster, T. L., Redding, R. E. and Kaempf, G. L.: A Skill-Based Cognitive Task Analysis Framework. Chipman, Shalin and Schraagen, (eds.) Cognitive Task Analysis. New Jersey: Lawrence Erlbaum, (2000) 135-146.
25. Kizlik, B.: Thinking skills vocabulary and definitions . From http://www.adprima.com/thinkskl.htm
26. Sloane, K. D. and Linn, M. C. Instructional Conditions in Pascal Programming Classes. R. E. Mayer (Ed.), Teaching and learning computer programming. Hillsdale, New Jersey: Lawrence Erlbaum Associates, (1988), 137-152.
27. Costa, S. e Moreira, M.: Resolução de problemas I: diferenças entre novatos e especialistas. Investigações em Ensino de Ciências, Porto Alegre, Vol. 1, No.2, (1996) 176-192.
28. Simon, D.P. e Simon, H.A.: Individual differences in solving physics problems. R.D. Siegler (Ed.), Children's thinking: what develops? Hillsdale, N.J.: Lawrence Erlbaum Associates, (1978) 325-348.
29. Larkin, J. H., McDermott, J., Simon, D. P. and Simon, H. A.: Expert and novice performance in solving physics problems. Science, Vol. 208, (1980) 1335-1342.
30. Larkin, J. H.: Enriching formal knowledge: A model for learning to solve textbook physics problems. J. R. Anderson (Ed.), Cognitive skills and their acquisition. New Jersey: Lawrence Erlbaum Associates, Publishers, (1981) 311-334.
31. Schunk, D.: Learning theories: An educational perspective (3rd Ed.) Upper Saddle River, New Jersey: Prentice-Hall, Inc, (2000).
32. Mestre, J. P.: Implication of research on learning. Physics Education, Vol. 36, No. 1, (2001) 44-51.
33. Neves, D. M. and Anderson, J. R.: Knowledge compilation: Mechanisms for the automatization of cognitive skills. J. R. Anderson (Ed.), Cognitive skills and their acquisition. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, (1981) 57-84.
34. Shiffrin, R. M. and Schneider, W.: Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. Psychological Review, Vol. 84, No. 2, (1977) 127-190.
35. Hanciles, B., Shankararaman, V. and Munoz, J.: Multiple representations for understanding data structures. Computers & Education. Vol. 29 No. 1, (1997) 1-11.

36. Brown, M.: Exploring algorithms using BALSA-II. IEEE Computer. Vol. 21 No. 5, (1988) 14-36.
37. Mendes, A. and Mendes, T.: VIP - A tool to VIsualize Programming examples. In Proc. of the EACT 88 - Education and Application of Computer Technology, EACT 88, Malta, October (1988).
38. Stasko, J.: Animating algorithms with XTANGO. SIGACT News. Vol. 23 No. 2, (1992) 67-71.
39. Levy, R. B., Ben-Ari, M., Uronen, P. A.: The Jeliot 2000 program animation system. Computers & Education, Vol. 40 No.1, (2003) 1–15.
40. Korhonen, A., Malmi, L., Silvasti, P.: 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, (2003) 48-56.
41. Kolling, M., Quig, B., Patterson, A. and Rosenberg, J.: The BlueJ system and its pedagogy. Journal of Computing Science Education, Special Issue of Learning and Teaching Object Technology, Vol. 12, No. 4, (2003) 249–268.
42. Naps, T.: Jhavé – Supporting Algorithm Visualization. IEEE Computer Graphics and Applications, Vol. 25 No. 5, (2005) 49-55.
43. Anderson, J. R. e Reiser, B. J.: The LISP tutor. Byte, Vol. 10 No.4, (1985) 159-175.
44. Song, J. S., Hahn, S. H., Tak, K. Y. e Kim, J. H.: An intelligent tutoring system for introductory C language course. Computers & Education. Vol. 28 No. 2, (1997).
45. Papert, S.: Mindstorms, children, computers and powerful ideias. New York: Basic Books, (1980).
46. Pattis, R.: Karel the Robot: A gentle introduction to the art of programming. (2nd Edition), John Wiley & Sons, (1981).
47. Brusilovsky, P.: Program visualization as a debugging tool for novices. In Proc. of INTERCHI'93, Amsterdam, 24-29 April (1993), 29-30.
48. Jehng, J., Shih, Y., Liang, S. e Chan, T.: Turtle-Graph: A computer Supported Cooperative learning environment. In Proc. of the ED-MEDIA'94. (1994) 293-298.
49. Cooper, S., Dann, W., Pausch, R.: Teaching objects-first in introductory computer science. In Proc. of the 34th Annual SIGCSE Technical Symposium on Computer Science Education, Vol. 35, No. 1, Reno, Navada, USA, (2003) 191-195.
50. Gomes, A. e Mendes, A. J.: "SICAS: Interactive system for algorithm development and simulation", in Manuel Ortega y José Bravo (Ed.), Computers and Education in an Interconnected Society, Kluwer Academic Publishers, January (2001) 159-166.
51. Rebelo, B., Marcelino, M. J., Mendes, A. J.: 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).
52. Myers, I. B. and McCaulley, M.H.: Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator. Palo Alto, CA: Consulting Psychologists Press, (1985).
53. Kolb, D. A.: Learning Style Inventory: Technical Manual. McBer and Company, Boston, (1985).
54. Felder, R. M.: Learning and Teaching Styles in Engineering Education. Journal of Engineering Education, Vol. 78, No. 7, (1988) 674-681.