Identifying Cognitive Abilities to Improve CS1 Outcome

Ana Paula Ambrósio, Fábio Moreira Costa, Leandro Almeida, Amanda Franco, and Joaquim Macedo
apaula@inf.ufg.br, fmc@inf.ufg.br, leandro@ie.uminho.pt, amanda.franco@ie.uminho.pt, macedo@di.uminho.pt

Abstract – **Introductory programming courses entail students' high failure and dropout rates. In an effort to tackle this problem, we carried out a qualitative study aiming to shed some light on the programming phase that is most challenging for students, in order to elicit the specific difficulties they experience while learning to program. In doing so, distinctive cognitive abilities, differentiating subjects in terms of the way they handle programming tasks, were detected. Such aptitudes are represented in three groups of students: those who learn easily, those who never seem to fully grasp what programming requires despite true effort, and those who experience a sudden insight, making them leap from a point were they had difficulties to another where they overcome them. By interviewing teachers and students, abstraction and sequencing elaboration were found to be the two core skills for programming. These results impelled us to consider the mental models' approach, concluding that there are very specific cognitive functions that are more favorable to learn programming and that are fostered by more adequate schemas of representing reality. Some conclusions involving Problem-based learning as a fit teaching methodology to overcome students' difficulties are also presented.**

Index Terms – Computer Science 1, algorithmic reasoning, programming learning difficulties, cognitive abilities.

INTRODUCTION

Generally speaking, introductory programming courses (CS1 courses) present challenges regarding student failure and dropout rates. To diminish these effects, several strategies have been adopted by different institutes in the organization and teaching of these courses. In the specific case of the Federal University of Goias (Brazil), we adopted the use of mobile, pen-based, computing technology and Problem-Based Learning in the redesign of our introductory computer programming course. In comparison with previous years, this approach achieved lower dropout rates and fewer grade failures.

Even though we obtained a significant improvement, the students that failed still represent a challenge that we must investigate. Our observations showed us that there were some students who, in spite of their many difficulties at the beginning, were able to make a leap at a certain moment and catch up with the rest of the class. Others, although committed and investing much time and effort, were unable to make that leap**.**

This study tries to identify the inherent causes of that phenomenon, from the perception of the actors directly involved. It differs from other approaches that try to relate success to demographic or cognitive characteristics of students obtained through the correlation of these characteristics and the grades obtained in introductory programming courses. Neither does it have the intention of isolating students who possess "aptitude". Its aim is to identify the phase in the programming process where students have the greatest difficulty, allowing us to focus on this aspect. We then make assumptions regarding the skills needed in that phase and imply them to be essential, acting as an important discriminating variable.

The study's starting point was the observation of students' behavior in the classroom. Based on these results, interviews with teachers and students were conducted by psychologists to refine which are exactly the difficulties in the process of learning to program and which skills are identified. Such perceptions were then used as guidelines to the application of a questionnaire to evaluate students' perceptions about the phase they considered the most difficult in programming.

Our focus was the cognitive variables involved in the task of learning to program. We must see that in cognitive psychology, learning and problem solving imply the use of cognitive abilities or functions; hence, we assume that programming must also make demands in this aspect. In fact, the compilation of our results allowed us to identify two main cognitive abilities associated to the difficulties presented by the students: the ability to carry out an abstraction of the problem at hand, and the ability to define a sequence of commands that allows the computer to solve the problem.

A theoretical research of methodologies associated to the identified cognitive abilities was used to propose strategies and tools that help to identify students at risk, and to suggest modifications to the existing teaching program as to enhance the acquisition of these abilities providing the students with mechanisms that may help them succeed.

OUR TEACHING METHODOLOGY

Problem-based learning (PBL) is "an instructional method characterized by the use of 'real world' problems as the context within which students learn critical thinking and

978-1-61284-469-5/11/$26.00 ©2011 IEEE

problem solving skills, and acquire knowledge of the essential concepts of the course. Using PBL, students acquire lifelong learning skills which include the ability to find and use appropriate learning resources" [1]. Even though the use of PBL does not necessarily mean an increase in grades, it has been verified that it fosters knowledge retention and enhances intrinsic interest in the subject matter [2]. Furthermore, it has led to recognized improvements in student programming skills related to abstraction and problem solving, and also in communication and argumentation skills, as well as in responsibility and peer support [3].

Mobile technology, in our case based on tablet PCs, enables a much more flexible classroom environment, where students and tutors are free to move around, carrying their permanently connected devices if needed. This is similar to previous initiatives involving the use of handheld devices in education, such as in [4], and is key to facilitate classroom rearrangement, interaction, experimentation, and access to external resources. In addition, ink-based computing presents students with a powerful tool for note taking and for expressing their creativity when working in the abstract reasoning associated with algorithmic thinking. Furthermore, the simple use of tablets is a stimulating factor, attracting attention to the course and contributing to engage students. It gives them more possibilities to collaborate, exchanging, evaluating and complementing each other's solutions to problems. Tablets also facilitate the implementation of the PBL method by giving students access to on-line information "at the tip of the pen", instrumenting the search for solutions and helping increase their proactivity and content retention. Thus, teachers move from an information providing position to a guiding position, focusing on teaching students how to think for themselves, stimulating logical reasoning and independence.

In our classroom experience, we use the PBL method [5] to introduce the concepts in the course syllabus as a series of open-ended problems, using a method adapted from Nuutila et al.[3]. Groups of four or five students work collaboratively to reach a solution to the proposed problems [6].

In order to contribute to this process, a mix of programming related tools have been used to help students think the problems abstractly and collaboratively. In the beginning of the course, we introduce a visual programming environment, which enables students to focus on the semantic aspects (logic) of the problem instead of worrying about syntax. We are currently using the SICAS environment [7] that allows the students to define executable flowcharts. Later on, we introduce a more traditional programming language, using the DevC++ environment. As students usually have a tendency to jump directly from problem definition to implementation, skipping the abstraction/algorithmic problem-solving phase, they are required to define a flowchart diagram describing the proposed solution before proceeding to implementation. Two traditional exams (at the middle and end of semester) are used for assesment purposes.

A first evaluation of our methodology was undertaken in 2009 [6], mainly by means of observations and surveys answered by students at different moments during the course of the semester. It was based on two classes of undergraduate CS students, totaling about 80 subjects. These students took the introductory computer programming course in the first semester of 2008 and in the first semester of 2009.

The evaluation concluded that the use of PBL promotes students' proactivity and that the necessary group interaction helps to develop communication and collaboration skills. Even though PBL was initially criticized by students due to the workload it imposed, the great majority of them believed it was a positive contribution to their learning process. They also believed that tablet PCs represent a valuable tool, not only for motivating students due to the innovative technology, but also due to their flexibility for collaboration and the sharing of ideas when compared to desktop and laptop computers.

Thus, the proposed methodology attained its goal of being motivating and stimulating from the start, engaging students and achieving lower dropout rates. Even though students did not obtain significantly higher grades in the written exams, the average overall failure rate (including drop outs and grade failures) was around 21%, as opposed to nearly 45% in previous years. The new methodology had a positive influence on students, not only from the academic/learning perspective but also from a personal perspective, making them feel more independent, proactive, responsible and prepared to work with peers.

Despite the advantages of the PBL methodology and its strategies and tools – previously presented –, the study undertaken recognizes the "need for further improvements on the methodology, targeting lower failure rates" [6]. This might be achieved by doing specific modifications to the used methodology, in order to detect beforehand those students facing trouble and in need of a more attentive assistance.

ANALYZING STUDENTS WITH DIFFICULTIES

Computer Programming is a highly complex activity, with subtasks related to different knowledge domains and a variety of cognitive processes [8], where a set of skills are valued, including: reading comprehension; critical reasoning and systemic thinking; cognitive metacomponents for problem identification, planning and resolution; creativity and intellectual curiosity; mathematical ability and conditional reasoning; procedural thinking and temporal reasoning; analytical and quantitative reasoning; as well as analogic, syllogistic and combinatory reasoning.

Generally speaking, Jenkins [9] conceives programming as a process aimed at the elaboration of a valid algorithm that will allow to elicit coding, being formed of various kinds of smaller and basic tasks:

"At the simplest level the specification must be translated into an algorithm, which is then translated into program code. In experienced programmers it is also possible to identify an intermediate process whereby the algorithm is mapped to something resembling a "recipe" for the program, based on previous experience."

Programming is a difficult undertaking, involving many different types of knowledge and abilities. According to Jenkins [9], transforming language into an algorithm is the most challenging step of programming.

In this respect, students entering computer science courses present very distinct behaviors. While some learn to program easily, others encounter huge difficulties. However, some are capable of surpassing these difficulties, as stated by Leither and Lewis [10]:

"Those who are mystified for weeks, and eventually make a quantum leap in understanding; and those who, despite extraordinary diligence and time-consuming effort, never succeed in making this transition." (in [11] pg. 39).

This same reality was observed in our group, and this is the reason why we undertook a qualitative analysis with students and teachers to help identify the conditions and moment in which the leap occurs, also intending that this might help to improve our success rates. Interviews were used to identify the factors they believe are decisive for learning to program, and the phase in the programming process where students encounter the major difficulties.

We have verified in the classroom that even when the students understand the problem and are capable of solving instances of this problem, they have great difficulty in translating this solution into a series of commands executable by a computer. These observations are corroborated by Winslow [12]:

"(…) novice programmers know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs. Even when they know how to solve a problem by hand, they have trouble translating the hand solution into an equivalent computer program."

The same author sets experts and novice students apart, considering that:

"(…) experts think in terms of algorithms and not programs. The actual translation of an algorithm into a working program is a task, not a problem. Presumably the algorithms allow them to concentrate on the important features of the solution and ignore the details which can be filled in later; in other words, it is a method for decomposing the problem solution into more manageable terms."

In our research we have tried to understand the reasons underlying these difficulties by listening to teachers and students. In this sense, we try to contrast the cognitive functions that may explain the difference between a good and a bad academic result in programming courses.

OUR STUDY

Students and teachers from the Federal University of Goiás participated in the study. Two teachers were interviewed. In order to obtain contrasted answers, we worked with two groups of students: one who learned easily and another who learned with difficulty despite evident effort. Both groups were selected by the teachers based on their observations and the student's grades. Nine students were interviewed: five with difficulty and four that learned easily. Of the nine, only one was female. Most were doing the course for the first time, only one was repeating. Two had previous contact with programming before the course.

The semi-structured interviews were conducted by two psychologists and a computer science teacher. Instead of focusing on motivational or learning style aspects, we tried to focus on the cognitive variables that teachers and students believe to be responsible for the learning difficulties.

The interviews were undertaken in one of three groups: the first with teachers, the second with students who face difficulties and the third with students who learn easily. During the interviews, we tried to focus the subjects' reflections on the skills needed to program, in order to infer which cognitive functions are involved in learning the curricula. In situations where the subject's idea was not clear, examples were asked for or another student of the group was called to help clarify what was being said by his/her colleague.

According to the teachers, the distinction between a group that learns easily and one showing difficulties that often lead to dropping out or failing the course is clear. They also identify a third group of students that have difficulty, but at a given moment are able to surmount their problems and advance in the course. They describe it as an insight that allows them to move from a situation where they were having difficulties to a position where they are capable of programming.

When analyzing these subgroups of students, the teachers believe there are two great factors, i.e., two cognitive abilities that are responsible for the student's results. The first cognitive ability relates to the ability of giving up a more holistic analysis of everyday problems in favor of a more analytical reading of such problems, being able to make a sequential planning. Almost in terms of a cognitive style, the student will have to leave the global and obvious, and pass to a more detailed analysis, where each solution is planned step-by-step, even though this may not be the usual manner of perceiving and solving problems in every-day life. In other words, students need a global vision of the problem and its solution, but it is equally important to have a vision of the parts that will compose the solution and that need to be organized and composed in a sequential manner. In their opinion, it is this shift in perception that allows them to see the whole based on the parts that will

allow the student to elaborate the algorithm. In the teachers words, when thinking this way, the student "no longer thinks as a person but starts thinking as a computer" taking into account the machine's limited comprehension. This is often referred to as algorithmic reasoning.

A second cognitive ability mentioned by the teachers is abstraction. It is expected that students will use it to pass from the concrete world to a more semantic and symbolic reality, formulating or representing a problem in a more abstract manner, less attached to the details and singular concrete elements. When a student faces difficulties while attempting to do this abstraction, he/she is not capable of moving from reality to a more generic language such as the notation used in programming. The ability to abstract is linked to the need of finding generic solutions that can be applied to several situations. Thus, when trying to find a solution to a given problem, the student is not constrained by specific data. On the contrary, he/she tries to abstract the "process" that can be applied to several situations from the solution. Often this involves breaking down the problem into subparts and identifying the relations between them.

In addition to the cognitive abilities discussed above, there are others that were mentioned. For example, some teachers believe that mathematical ability is important for programming. However they believe that the positive correlation between them is due to more basic cognitive functions that would be common to both domains. Thus, when developing a "mathematical logic" the student acquires structures or cognitive abilities that facilitate or promote learning to program.

We will now refer to the students' perceptions regarding the skills implied in programming. Starting with an analysis of the students with difficulty, it is to note that this subgroup seems to pass automatically from reading to implementation, skipping the definition of an algorithm. If we assume that the solution of a problem involves three phases – input, processing and output –, these students are jumping from the first to the third phase, without the necessary processing of the coded information and the task's purpose.

For them, reading is not a problem; any obstacle is due to the ambiguity of natural language. On the contrary, the language used to program is difficult. They mentioned aspects such as passing parameters and functions. When questioned about the intermediary phase and the need to undertake some procedures between the reading and implementation phase, they simply seem to disregard this process as trivial. However they encounter great difficulty in finding an abstract solution to the problems in hands.

On one hand, these students believe that the solution must be mentally structured in their heads, and then implemented. However they jump to the implementation before having a complete solution defined. In fact, they go for an initial idea and try to implement it, programming through trial and error, without a prior global vision of the solution. When questioned about the difficulties of finding an abstract solution, they are not able to elaborate their answers: they feel the difficulty but do not know why or where it occurs. They simply disregard it and thus pass directly to the implementation phase.

On the other hand, in the interview conducted with the students who learn easily, the importance given to the intermediary phase is clear: they believe that thinking about the solution before implementing it is crucial. They globally highlighted the importance of understanding the problem's logic, reason why they first look at the problem as a whole and try to identify its focus. In the words of one of the students: "First you have to understand the problem and search for solutions. Think in different ways and choose the solution that you think is best". If they are not able to find a solution, they break the problem down and then identify the relations between the parts.

These students believe that the main difficulty of programming is in the reasoning needed to find generic solutions. According to them, many of their colleagues do not even try to arrive at an abstract solution. One of the students said he usually "attacks" the problem as a whole, but when faced with difficulty he divides the problem in parts. They all agreed that visual tools that help them think about a problem are welcome. For this reason they like working with flowcharts: "We need to put the problem on paper. Doing it all in your head is not possible". By breaking down the problem they can work with smaller parts and see how they relate.

It is important to stress that these students also believe that mathematical dexterity is important as well as creativity, even though they state that programming is basically reasoning.

Summarizing the results, we believe that the students that learn easily develop a theory of the problem and of the solution, within the definition of Naur [13], which contributes to better programs. In this sense, programming is a task where the solution algorithm is mapped directly to a programming language. This difference of strategies, which may translate into the existence or lack of subjacent cognitive skills, may be responsible for the different perceptions regarding the difficulty of the programming phases. While students with good results evaluate the initial semantic phases as more difficult, students with weak results attribute greater difficulty to the implementation phases. It is interesting to observe that, in the literature, experts and novices are said to structure their knowledge differently: while experts focus on the commands' semantics, novices focus syntax [8][14], much as verified by the interviews.

Finally, as to try to capture the students' perception of the difficulties found in the different processes of programming, as well as the phase they found to be the most challenging, we asked students to answer a questionnaire where programming was divided in 8 phases:

1. Reading and understanding the problem;
2. Solving an instance of the problem by hand;
3. Generalizing the solution;
4. Elaborating an algorithm that solves the problem;
5. Simulating the algorithm;

6. Translating the algorithm into a programming language;
7. Compiling;
8. Testing.

The students were asked to mark the level of difficulty they attributed to each phase using a six point Likert Scale, corresponding 1 to a minimum level and 6 to a maximum level, and to classify these phases according to an ascending order of difficulty. They were also asked to evaluate themselves, grading their performance in CS1.

Eighteen surveys were answered, not necessarily by the same students that were interviewed. However, they were all from the same classes.

Table I presents the level of difficulty attributed to each phase by the students. The last row contains the grade the student gave to his/her programming knowledge. The phases considered more difficult are exactly those related to abstraction and generalization: Solving an instance of the problem by hand, Generalizing the solution and Elaborating an algorithm that solves the problem. Reading and understanding the problem (phase 1) presented some difficulty, confirming the teachers' perception regarding the student's ability to read and understand the problems' specification. The phases related to syntax had lower median values.

TABLE I
LEVEL OF DIFFICULTY ATTRIBUTED BY STUDENTS TO THE DIFFERENT PROGRAMMING PHASES

Phase	Student's Answers																		Median
1	4	2	3	1	5	2	2	3	3	4	4	4	4	2	3	5	4	3	3
2	6	3	3	1	4	4	1	1	3	4	5	4	3	4	4	2	4	4	4
3	4	4	5	1	6	4	4	2	5	4	2	2	3	4	4	2	3	3	4
4	4	4	4	3	3	2	4	6	3	4	1	1	3	4	4	1	4	4	4
5	1	6	3	2	3	2	2	4	3	4	2	2	2	2	4	1	3	4	2,5
6	5	4	4	2	4	2	3	2	1	4	3	2	2	1	2	1	3	5	2,5
7	2	2	3	3	1	1	1	1	1	4	1	1	2	1	2	1	4	4	1,5
8	4	1	4	5	2	1	1	2	4	4	6	5	2	3	2	1	3	2	2,5
Grade	4	2	4	3	4	3	-	1	6	4	4	4	4	5	5	-	4	4	

A heterogeneous view of the programming difficulties was verified when the students were asked to classify the programming phases in ascending order of difficulty. On one hand, phase 7 was considered the easiest one, followed by phase 8. No one considered phases 2 and 3 as the easiest (Figure 1). On the other hand, while 25% of the students considered phase 3 the most difficult, another 25% considered phase 8 as the most difficult. No one considered phases 6 and 7 as the most difficult (Figure 2). A special comment should be made about phase 8 (Testing): 25% of the students considered it very easy, while 25% considered it very hard. This may be due to different understanding about the tasks involved in this phase. Previous observation had shown that students find it very hard to define test cases. However, once they are defined, their use to verify if the program yields correct answers is easy.

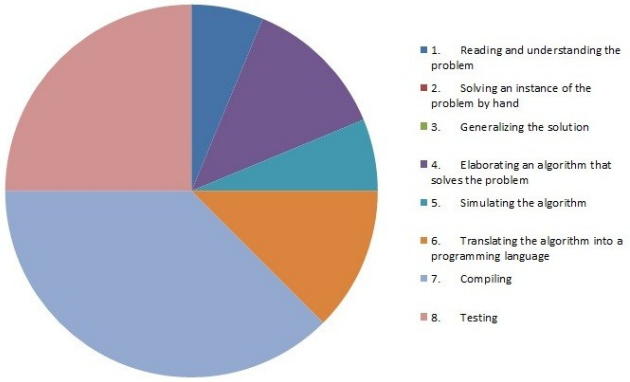

FIGURE 1
PHASES CONSIDERED EASIER BY STUDENTS

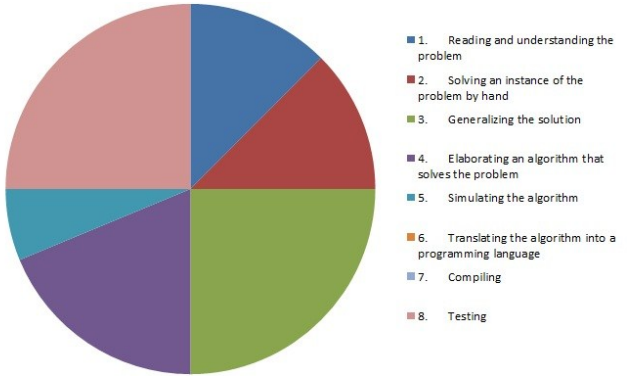

FIGURE 2
PHASES CONSIDERED MORE DIFFICULT BY STUDENTS

In addition to the analysis of the difficulties associated to the different programming phases, the questionnaire also asked about the difficulties encountered in specific CS1 curricular contents. It also enquired about their study habits, and the resources they find more helpful. Declaration and manipulation of variables were considered the easiest content. Recursion was the most difficult, followed by the division of a problem into parts that can be reused and interpretation of the error messages. Students find it easy to research for helpful material in the Internet, but have difficulty studying at home. The resources they believe to contribute the most to their learning process is Moodle, followed by examples of program codes. Chats with teachers and TAs were considered the less helpful.

DISCUSSION AND CONCLUSION

We have verified in introductory computer programming courses that a group of students has a lot of difficulty even when demonstrating to have interest and effort to learn. Some within this group are capable to surpass the problem and succeed; others, nevertheless, are not.

Two cognitive abilities were identified as key to explain the difficulties the students encounter when learning to program: abstraction and command sequencing. These

cognitive abilities agree with those encountered by Lee et al. [15] for Computational Thinking: abstraction, automation, and analysis, where automation corresponds to command sequencing.

We infer that underlying the abstraction difficulties are the student's problem solving abilities. Problem solving is composed of Analysis and Synthesis. Analysis is the ability to break a problem down into its subparts and look at them individually so that the problem can be more easily understood and treated. Synthesis is putting the subparts together after they have been treated individually as to obtain a solution to the original problem that was being tackled.

Differences on problem solving abilities are due to cognitive processes and mental organizations [16]. Some characteristics shared by good problem solvers include [17]: skill with analogies, reasoning, critical thinking, perception, memory and creative thinking. They additionally have good reading comprehension skills and possess knowledge of different approaches that can be used to solve a problem. Efficient problem solvers have knowledge that is organized and rich in variety. They focus more on the structural features of a problem and not just the surface features.

This is exactly what is done by the best achieving students. They analyze the problem and when they are not capable of finding a solution they break the problem down. Contrarily, the low achieving students are not capable of doing this: they focus on reading the problem superficially, without reaching its core meaning.

The identified difference of approach used by proficient students and students facing difficulties in programming may be linked to their mental models. "Mental models" define a concept in the field of Cognitive Psychology referring to the cognitive format in which information regarding reality external to the subject is apprehended and organized [18]. Considering them to work as a data base containing knowledge with which we interpret reality and that is itself transformed through the apprehension of further and deeper knowledge, mental models can be perfected [19]. There are different levels and quality in knowledge apprehended according to the mental models detained [20], reason why we believe there should be efforts to promote and strengthen the best possible fit between the students' mental models and the curricular contents they are expected to learn and use.

Mental models have been studied from different programming perspectives. According to Mayer [20], students that do not have a mental model of how a computer stores and manipulates data in memory, have greater problems understanding programming language commands. Other authors have focused on the differences between novice and experts: while novices see programs as sequences of commands, experts group commands in schemas that represent functionalities. Cañas, Bajo and Gonzalvo [14] verified that students have different mental representations of computer programs that may be based on syntactic or semantic aspects. Denhadi [21] proposes that students that have a consistent mental model of variable manipulation, have higher success rates.

Linking this concept to our study, it is viable to infer that students who learn easily may have mental models, acquired through other subjects such as Math, Science or English, that might ease learning to program. This would explain why good grades in these courses present a positive correlation with programming. The students that have difficulty might not have these mental models, or have less adequate ones, in which they can anchor newly acquired knowledge, reason why they would need to develop or correct them. Those that are able to do that can succeed, but those that don't will fail. In this sense, we propose the introduction of activities that foster the development of mental models that are fit to the cognitive skills needed in programming, thus helping the students succeed in the course.

While problem based learning tries, by definition, to promote problem solving abilities, we have verified that it is not enough when dealing with the students that present greater difficulties. Building on the mental model researches, we propose activities to be included in our methodology that will promote developing abstraction and command sequencing abilities. These activities are made available to those students that feel they need it and are willing to do them.

Our proposal is that these activities could be undertaken during tutoring sessions under TA supervision. In fact, despite some constraints deriving from the nature of this construct, it is possible to assess the characteristics of a student's mental model. This can be done by using strategies of behavior observation and self-evaluation reports (Moreira, 1996). Such information could serve as a starting point for intervention efforts.

Activities include acting out how data manipulation is carried out by the computer, and reading programs developed by others in order to alter them to include, modify or remove certain functionalities. Another activity that is being proposed is based on the construction of programs using pre-defined schemas, much in the sense that experts see schemas that represent functionalities. In this case, students have a set of cards that represent these high level functionalities and they have to select those that contain the functionalities existing in the problem. These functionalities are very general and can be seen as functions or procedures. Examples of these functionalities include input data and classification of an array. They then have to organize and link these cards to form the global solution. In a second moment, they must look into each of these functionalities and redo the same process until they get to a complete algorithm.

These proposals are being implemented and monitored to verify their contribution to the learning process of students with difficulty. In addition, we want to deepen our understanding of the cognitive aspects that are most relevant to the process of learning to program. In this sense, psychological tests will be applied to the two groups of

students, those that learn easily and those with difficulty, and the results will be analyzed in order to refine the proposed activities.

REFERENCES

[1] Duch, B. J., 1995, "What is problem-based learning?" *About Teaching*, 1995(47).

[2] Norman G. R. and Schmidt H. G., 1992, "The psychological basis of problem-based learning: A review of the evidence." *Academic Medicine*, 67(9), pp. 557–565.

[3] Nuutila E., Torma S. and Malmi L., 2005, "PBL and Computer Programming – The Seven Steps Method with Adaptations." *Computer Science Education*, 15(2), pp.123–142.

[4] Norris C. and Soloway E., 2004, "Envisioning the handheld-centric classroom." *Journal of Educational Computing Research*, 30(4), pp. 281–294.

[5] Schmidt H. G., 1983, "Problem-based learning: Rational and description." *Medical Education*, 17, pp. 11–16.

[6] Ambrosio A.P. and Costa F.M. "Evaluating the impact of PBL and tablet pcs in an algorithms and computer programming course." In SIGCSE '10 Conference, March 2010, *Proceedings of the 41st ACM technical symposium on Computer Science Education*, New York, pp. 495–4

[7] Antunes, R. *Ambiente de apoio à aprendizgem de programação Web utilizando PHP*. Master Thesis, University of Coimbra, 2005.

[8] Pea, R. D. and Kurland, D. M. "On the cognitive prerequisites of learning computer programming", *Technical Report No.18*. 1984, New York: Bank Street College of Education, Center for Children and Technology.

[9] Jenkins, T., "On the difficulty of learning to program." In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. 2002, Loughborough: University United Kingdom, pp. 53-58**.**

[10] Leither, H. E. and Lewis, H. R., 1978, "Why Johnny can't program: A progress report." *SIGCSE Bulletin, 10* (1), pp. 266-276.

[11] Robins, A., 2010, "Learning edge momentum: A new account of outcomes in CS1." C*omputer Science Education, 20* (1), pp. 37-71.

[12] Winslow, L. E., 1996, "Programming pedagogy: A psychological overview." *ACM SIGCSE Bulletin, 28* (3), pp. 17-25.

[13] Naur, P., 1985, "Programming as theory building." *Journal of Systems Architecture: The Euromicro Journal*, *15* (5), pp. 253-267.

[14] Cañas, J. J., Bajo, M. T. and Gonzalvo, P., 1994, "Mental models and computer programming." *International Journal of Human-Computer Studies, 40* (5), pp. 795-811.

[15] Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. and Werner, L., 2011, "Computational Thinking for Youth in Practice", *ACM Inroads*, 2(1), pp. 32-37.

[16] Chi, M. T. H. and Glaser, R., "Problem solving ability." In R. Sternberg (Ed.), 1985, *Human Abilities: An Information-Processing Approach,* San Francisco: W. H. Freeman & Co., pp. 227-257.

[17] "Research-based Support for Mathematics Teachers", 2000, *Kansas State Department of Education,* http://www.ksde.org/, Accessed April 2nd, 2011.

[18] Gentner, D. "Mental models, Psychology of." In N. J. Smelser, & P. B. Bates (Eds.), 2002, *International encyclopedia of the social and behavioral sciences* (pp. 9683-9687). Amsterdam: Elsevier Science.

[19] Johnson-Laird, P. N., 1995, *Mental models* (6th ed.). Cognitive Science Series, 6. Cambridge, MA: Harvard University Press.

[20] "Représentation de la connaissance." 1996, http://tecfa.unige.ch/staf/staf9597/beltrame/STAF11/concepts.html, Accessed April 2nd, 2011.

[21] Mayer, R.E., "The psychology of how novices learn computer programming." In E. Soloway, & J.C. Spohrer (Eds.), 1989, *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[22] Dehnadi, S., "Testing programming aptitude." In P. Romero, J. Good, E. A. Chaparro, & S. Bryant (Eds.), 2006, *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group*. University of Sussex, pp. 22-37.

AUTHOR INFORMATION

Ana Paula Ambrósio, Computer Science Institute, Universidade Federal de Goiás, Brazil, apaula@inf.ufg.br.

Fábio Moreira Costa, Computer Science Institute, Universidade Federal de Goiás, Brazil, fmc@inf.ufg.br,

Leandro Almeida, Educational Psychology, Universidade do Minho, leandro@ie.uminho.pt,**,**

Amanda Franco, Educational Psychology, Universidade do Minho, amanda.franco@ie.uminho.pt

Joaquim Macedo, Department of Informatics Engineering, Universidade do Minho, macedo@di.uminho.pt