A Tutor to Help Students to Build Abstractions in Programming Logics Classes

M.M.Mattos,Dr.

Abstract—Educators are constantly seeking new ways to facilitate learning and maintain the attention of their students. This article describes the problem that affects students in the first steps in programming logic classes. The problem is characterized, and a new approach to solving it is presented using expert systems techniques, case-based reasoning and software engineering concepts. An expert system tool prototype was developed to support this effort. This prototype is described and some conclusions are presented.

Index Terms— Software Engineering, Computing Education, Passive/Active Learning.

I. INTRODUCTION

A ccording to [9], “studying is an activity that consists of assimilating a new situation in the action outlines used in other situations. Each new learning supposes new knowledge. Studying then, means trying new strategies of behavior, and then transferring or generalizing those behaviors that have previously been incorporated through experience”.

As well stated by Michael Byrne [3], educators are constantly seeking new ways to improve instruction, facilitate learning, and maintain the attention of their students. The ability of computers to store vast quantities of information and to simulate environments and conditions that would otherwise be unavailable makes the intriguing possibilities for use as instructional aids.

Along with the growth in the use of computing technologies applied to education, it is important to point out not only the contribution of existent computing resources, but also the need to study the ideal form of application. The use of this technology in instruction raises many questions, as would the incorporation of any new instructional aid. According to Lawrence [10], one question, which has frequently arisen, is whether such techniques would be superior to transparencies leaving other forms of access aside.

This work presents a problem that affects students in the first steps in programming logic classes, and then discusses the graphic approach (largely used in tools to assist them) versus the textual solution described here. After presentation of prototype tool characteristics that has been built, there is a discussion of its application in teaching software engineering concepts in the first steps in Computing Courses.

II. THE PROBLEM'S CONTEXT

Beginning undergraduates in Computer Science at FURB – The University of Blumenau - Brazil, usually have difficulties in the Algorithms class, where the main objective is to introduce basic concepts related to programming logic. This difficulty stems from several reasons.

The first is related to the students’ profile: most of them are starting their undergraduate studies as the next step in an education chain. They have abstract knowledge of the sciences (mathematics, physics, biology, etc.), but when faced with text descriptions of the assignments posted for this introductory class they usually have problems identifying and grasping the principal goal and extracting all the information needed to start solving the problem. The second aspect refers to a lack of professional experience in business areas, upon which several assignments are built. Lastly, some students, while grasping the main goal, are not able to break the problem down into small steps.

In our experience in teaching this class, we observed that students could be divided into two groups: those that had realized and overcome the initial difficulties, and those that hadn’t. Moreover, we observed that many in the second group, when induced to think on the subject (through
addressed questions), were able to describe the solution intuitively. Starting with analysis and observation of the inductive process, work began during the first semester of 1998 to formalize a methodology of this process, with the objective of guiding the student through the problem solving process. This process was used throughout the semester and at the end the students evaluated the proposal as being very useful, mainly because it established a rule-of-thumb (according to some students), i.e., a formal procedure to be followed in solving those exercises.

Initial results led to refinement of the original proposal and to the development of an expert system, which, starting with informal knowledge of the inductive process, allowed the construction of a software tool that enabled the student (in the teacher's absence), to develop the process of learning the basic concepts of the discipline.

Another important aspect to consider is that taking the expert systems literature as a starting point, we often found references to the application of this technology in the solution of such complex problems as medical diagnosis, market analysis, credit analysis, and so on. It was observed however, that the employment of expert systems in the computing area is not greatly emphasized. Finally, the programming logic course is the first stage towards forming a professional profile in the computation area. Bad habits acquired in the first phases can be difficult to correct but if some formalism is applied in the beginning, it will certainly contribute to the formation of better professional skills.

III. SOFTWARE VISUALIZATION

The use of animation to teach programming and how algorithms work is called algorithm animation and an instance of software visualization in which images, graphics, and animation are used to illustrate computer algorithms, programs, and processes.

Algorithm animation actually serves two fundamental purposes as an instructional aid: it provides a concrete depiction of the abstractions and operations inherent in an algorithm or program, and it portrays the dynamics of a time-evolving process. For these reasons, educators and algorithm animation system developers have hypothesized that algorithm animations can become valuable instructional aids. Unfortunately, this hypothesis is largely based on intuition and anecdotal evidence [3].

A number of algorithm animation systems have been built over the last ten years. Just a few of the better known and more widely used systems include BALSA [1], Tango [11], Zeus [2], and AACE [7]. The purpose of these systems is to allow dynamic visual representation of programs or algorithms. Such representations may be useful to appeal to the power of the human visual system. Since computer programs may be unclear in textual format, it is hoped that the graphic animated format will aid understanding. Many of these systems have consequently been used as instructional aids [10].

According to Colleen [8], a number of prior studies have found that using animation to help teach algorithms had less beneficial effects on learning than hoped. Those results surprise many computer science instructors whose intuition leads them to believe that algorithm animations should assist instruction. Their work reports a study in which animation was used in more of a “homework” learning scenario rather than a “final exam” scenario, concluding that: “Clearly, the use of computer multimedia technologies such as animation is growing throughout the educational community. Unfortunately, our understanding of how these technologies can be best used is lacking.”

They conclude with a citation from Mayer that affirms: “The potential for computer-based aids to learning environments remains high, although the current contribution of technology to pedagogic innovation is frustratingly low. Instructional development is too often based on what computers can do rather than on a research-based theory of how students learn with the technology. In particular, the visual-based power of computer technology represents a grossly underutilized source of potential educational innovation”.

Our choice in adopting a text-based tool wasn’t because of the aspects related above but mainly as a function of our experience in developing a text based set of rules that could establish one guided way for students to learn programming logic concepts more easily. In a further step in this project, we will probably consider using a graphic tool and applying animation concepts to enhance it.

IV. SOFTWARE ENGINEERING CONCEPTS

According Wilson [12], the importance of correctly documenting requirements has caused the software industry to produce a significant number of aids to the creation and management of the requirements specification documents and individual specifications statements. However, very few of these aids assist in evaluating the quality of the requirements document or the individual specification statements.

This situation has motivated the Goddard Space Flight Center’s (GSFC) Software Assurance Technology Center (SATC) to develop a tool to provide metrics that NASA project managers can use to assess the quality of their requirements specification documents and to identify risks that poorly specified requirements will introduce into their project.

The tool searches the document for terms that SATC has previously identified as quality indicators and the reports generated are used to identify specification statements and structural areas of the requirements specification document that need to be improved.

It’s important to point that the tool doesn’t attempt to assess the correctness of the requirements specified but to analyze the structure of the requirements document and individual specification statements and the vocabulary used to state them requirements.

The same philosophy has been used in the current project.
The idea is to conducte the student in the process of knowledge discovering in how to understand the specification statements and at the same time learning how to do it in a formal an ordered way.

V. USING ARTIFICIAL INTELLIGENCE CONCEPTS

According to Giarratano [6], knowledge can be classified as: a) procedural: knowing how to do; b) declarative: knowing that something is true/false; and c) tacit or unconscious: knowing what happens, but not how it does.

In the specific case of the proposed problem, it is frequently observed that starting with the solution of a certain number of exercises, some students easily discover the mechanism of how solutions are built. Others, however, who don’t discover this new form of thinking, walk around in circles, assuming trial and error tactics to try and guess the correct answer. Naturally, this leads to failing the discipline.

This act of discovering the correct way, if analyzed from the students’ viewpoint, could be considered tacit knowledge since, as is usual with those who learn by doing, they have difficulties in explaining to their colleagues how they organized the reasoning that led to the solution of the problem. Analyzed from the teachers’ viewpoint, this knowledge is procedural, i.e., there is a known sequence of steps that have to be followed in order to build a logical solution. Another type of knowledge involved here may be classified as declarative, i.e., analyzing the solution and verifying if it works. Based on previous experience, it is known that a certain sequence of steps used for some problems, will probably lead to a solution that is incorrect in certain situations. This context established the challenge of finding a solution, using expert systems techniques.

VI. METHODOLOGICAL ASPECTS

According to Durkin [4] there is a series of stages that should be followed in the development of projects of expert systems, applying an incremental life cycle model of systems development. Using this model as a reference, we describe below activities that were executed in each phase during the development of this project.

The first aspect was related to acquisition of knowledge. As mentioned previously, once the induction process showed good results the next step was to identify which questions and in what sequence they could be presented in order to guide the students towards correct solutions to the proposed exercises. Eight main rules were identified, which the students called the “8 steps methodology”. In this first version the students had to answer this sequence of questions for each proposed exercise:
-What are the known variables? - What information can be obtained starting from the statement of the problem?
-What has to be calculated (or executed)? - Identification of the application’s range or context;
-What are the unknown variables? - Variables for which there is no information in the statement;
-What needs to be entered (typed)? – What does the user need to key (mainly from the keyboard) to the application so that it can accomplish its functions;
-What needs to be presented (printed)? - What should the application present as processing results;
-To build a sketch of the solution - to identify macro-steps which could be the strategy to for solving the problem;
-To build a flowchart - to elaborate a flowchart, using graphic notation to represent the solution’s logic;
-To construct a step-by-step approach - to relate all evolving variables, beginning with commands and following the logic flow depicted in the flowchart.

Consequently, after constructing a step-by-step test approach and verifying it as correct, the solution is ready to be coded in some programming language.

Although this strategy led to easier problem solving, the methodology described above still needed refinement because there were obviously many hidden steps between initial phases and the last ones. In spite of this, a large number of students who used this strategy began to better understand the context of applying it. This because the first step in solving a problem understands it.

The next step was to try to refine the more obscure aspects previously mentioned. Through interviews with the specialist, and using the exercises lists the students used to receive, knowledge was refined to where a kind of spreadsheet, constituted of 28 columns, was obtained.

In this spreadsheet, besides the questions sequence already identified, classes of related questions were also identified, in such a way that, depending on the answers obtained, only a subset of the same needed to be answered in order to reach the final objectives.

Finally the knowledge acquired was represented in the spreadsheet verified that there were subjects that demanded yes/no answers; subjects that demanded textual answers; situations where some orientation was necessary to the guide the student to the next step and, situations where it was necessary to give a feedback about decisions that were taken, in such a way as to place them in the context of a solution.

Starting with this information, a decisions’ tree was built. This tree represents the specialist's knowledge in terms of procedural knowledge, as mentioned previously. The strategy adopted to acquire the student's knowledge is described below. A kind of top-down development was adopted, in which the development of the solution occurs by successive refinements. Whenever there is a need, a new refinement step is accomplished in order to obtain the desired level of specification that actually solves the problem.

To support this, a set of rules was built that allows a kind of sailing through the decision tree’s nodes. While the nodes are being visited repeatedly, new facts are being generated in the work memory of the expert system engine. These new facts fire rules that, once executed, generate a new auxiliary tree (denominated log tree). This tree registers the user's answers, and establishes a temporal sequence in which the answers are being created.
After each pass over the decisions’ tree the student receives a feedback screen of the context delineated by him up to that moment. Depending on the necessary level of refinement, the system automatically begins a new step of refinement if it detects that some structure still requires more information. The tool continues to present new questions to the student in such a way that the student is being led further and further towards a complete description of the problem’s solution.

This strategy allows tacit acquisition of knowledge on the part of the student, starting with the declarative knowledge of the expert. In other words, the student knows how to answer yes or no to the questions presented to him. The order of the questions is established beforehand in the decisions’ tree. What the student doesn’t know how to do, to explain how to solve the problem, is automatically being stored in the auxiliary tree (log) in such a way that, at the end of the process, it is possible to infer the macro-solution of the problem through a union of the several knowledge types involved in the process. The systems architecture is presented in figure 1.

Once the first prototype was built and validated by the students, a new version started to be developed starting from the evaluation given by students. This new version was built in java and use Case-Based Reasoning concepts. It has added the possibility to verify quality aspects over the solution given by students by comparing the solution’s profile given by the student with the system’s profile stored in a knowledge base.

When a new problem is posted, the system doesn’t have the solution so, the solution given by the student is taken as the possible correct one. As time passes and new students run the problem, the system starts to learn the process, and to compare the new solutions with those stored in its base. So, after a time, the system can better help the student in to find the approximate correct solution to the problem.

Both prototype versions have been ported to Delphi and Java and there is an experimental version available to download in http://www.inf.furb.br/~mattos/tools. This new version implements some new constructions such as repeat-structures and embedded if-structures. Figures 2 and 3 presents some screens to give an idea about this new version. Figure 4 presents the last version including a module to animate the flowchart execution. This module enables the student to test the solution produced by induction in the former steps. Unfortunately this version has not yet been translated to English.

VII. CONCLUSION AND FURTHER WORK

It can be considered that the prototype’s implementation could be used to validate the main objective. Initially defined as the development of a software application that could be used as a support tool to assist in the learning of programming logic.

One important aspect to be considered is that this work doesn’t introduce a new methodology to teach programming logic classes but introduces a new way to apply those well-known methods of teaching this class.

Another, but not least important aspect that must be stated is related to the fact that this proposal doesn’t lead the student in the sense of generating the correct solution for a certain problem. This is not the main objective of the tool. If the student answers incorrectly (the term incorrectly in this context being relative), the system will present a sketch of an agreed solution based on that answer. In doing so, the tool doesn't eliminate the previously described philosophy of "8 step methodology ".

It can be classified simply as the requirements analysis phase in a traditional software life cycle. As a final product we can obtain a sketch of a solution.

This will still have to be transformed into a flowchart, which should be evaluated for quality through the use of the step-by-step approach mentioned previously. Only after that, can the solution be considered correct. Therefore, the most important aspect to point out is the fact that this solution allows the student to systematically learn how to think in terms of steps to solve the problems, and also to learn how to validate his solution.

Naturally, question as to the efficiency of the solution will be developed in time, when the student’s level of maturity is ready to allow this kind of discussion. In our current work, we are porting the CLIPS implementation to a Jess tool (java implementation of CLIPS engine), in such a way as to enable us to build a user-friendlier interface and to expand the current version. Another thread that we are working on is implementation of the rules mentioned in section V of this work related to identifying during the process the use of those qualitative words that could indicate better or worse descriptions and thus relate the results to a better or worse solution by the student.

ACKNOWLEDGMENTS

This work is supported by FURB – University of Blumenau. Our thanks to the graduation students: Luiz Heinzen, André Gubler, Gilson Klotz, Gilberto Freitas and Alisson Appio.

REFERENCES

[8] Colleen, K, Stasko, J. and Ashley, T. Rethinking the evaluation of algorithm animations as learning aids: an observational study. Graphics,


Fig.1 General architecture

Fig.2 Questions presented to student

Fig.3 Final answer to assignment

Fig.4 Animating the flowchart produced from the final answer

II Workshop de Tecn. da Inf. aplicada ao Meio Ambiente – CBComp 2004
Inteligência Artificial