Towards the development of a cognitive model of programming; A software engineering proposal

Des Traynor, J. Paul Gibson
Computer Science Department
National University of Ireland, Maynooth
dtraynor@cs.may.ie, pgibson@cs.may.ie

Keywords: POP-I.A. Learning to Program, POP-II.B. Program Comprehension, POP-III.B. Java, POP-IV.A. Evolutionary Prototyping, POP-V.B. Observation, POP-VI.E. Computer Science Education Research.

Abstract

Computer Science, particularly programming, is regarded as a very difficult subject to study and also to teach. Unlike other academic areas such as linguistics or physics, there is very little pedagogical research in the area. It is important to first understand how students learn, before attempting to teach them. The recent increase in the use of e-learning technologies for teaching computer science may be compromised by this lack of this fundamental knowledge. We propose the use of student profiling techniques combined with adaptive learning technologies to detect learning patterns and traits in students, thus recording how they learn. This empirical evidence can be used to reason about the development of a cognitive model of learning programming, potentially alleviating the aforementioned problem. Strong software engineering practices will control the development of this system and all experiments will be carried out using solid scientific methods.

Introduction

Many computer science students find programming extremely difficult to learn (Mc Cracken et al 2001), likewise, many of their lecturers find it a very difficult skill to teach (van Roy 2003). Research into the teaching of computer science is an overlooked area, despite being one of huge importance. With the recent boom in both e-learning and educational technology, many efforts have been made to move these advancements into computer science education. However, this effort may be in vain, as there is very little exploitation of pedagogical knowledge in the computer science domain. It is not well understood how people learn to program or to problem solve; as a result, when these efforts fail it is not necessarily the technologies that are at fault, the problem is simply the lack of the required background knowledge. This lacuna in computer science research has been previously observed (Holmboe 2001) and will, we believe, receive further highlighting as modern e-learning tools fail in the area of computer science education.

Traditional teaching methods do not adapt well to the domains of coding and problem solving, as these are skills best learned through experience. A difficulty that often occurs within traditional teaching methods is that students lack the specific problem domain knowledge to understand the solution. For example, students who have never heard of the Fibonacci numbers (Stojmenovic, 2000) do not appreciate a recursive function to print them out. These students tend to confuse problem solving with coding, and as a result fail to see clearly the relation and dependency between the two. This "abstract nature of programming tasks" has been referred to previously (Dunican, 2002) as a reason why students find programming concepts difficult to grasp.

This paper outlines how the development of a cognitive model will be attempted through the use of student profiling techniques combined with results obtained in an adaptive learning environment. It is believed that if the learning patterns of students can be observed through such a method, useful conclusions about the learning patterns of students can be obtained; e.g., what concepts should be taught together, what are the pre-requisites before certain topics can be covered sufficiently. The next section discusses what is required to attempt to develop and understand such a cognitive model.

Section three details the development methodology that will be employed while section four gives details of the desired implementation. The final section discusses the possible applications of such a cognitive model and details their potential impact on educational technology.

Requirements for Developing a Cognitive Model

To develop a cognitive model of how students learn programming, we must examine the manner in which they learn. This will be achieved through asking a sample of first year students to undertake several tutorial tests in an adaptive learning environment. First year students have been chosen as it is in the earliest stages of learning that students will exhibit the learning patterns that we hope to study. Profiles will be developed for each student representing their knowledge in particular areas, as interpreted by the system. The student profiling in this system will be performed by processing the students' history of tutorials taken and performance in tests to estimate their ability at programming. This estimation will be given assigned a value, so that the students' perceived ability can be monitored and their improvements noted.

The most basic student profile would consist of a single binary value representing if the student has 'sufficient programming ability', an extension of this simple profile would be a set of boolean values corresponding to whether the student has 'sufficient programming ability' in various programming concepts. This simple profile model however does not carry sufficient information regarding a students' actual ability within the continuum of learning (it is of course, possible to map a continuum of integer values onto a set of binary values, but this would be an impractical and naive approach to building such a profile model, where the model structure should give some insight into the structure of the learning process itself). Similarly, if a profile was chosen to provide the maximum information, it would carry the entire students' record in an uncompressed form, which would be computationally infeasible to use as a profiling mechanism. This trade-off between computational performance and semantic depth is dealt with in the following example, where we propose using values (from 1 to 5) to represent a student's ability for four programming concepts. Whilst the following sample profile is still clearly lacking in information, it does show that realistic student profiles could be evolved to be both accurate and practical.

The sample profile will be a function mapping a set of 4 integers to their corresponding concepts that they represent. A five point scale is chosen for this example, where 1 represents no knowledge, and 5 represents sufficient knowledge. For example the profile model (variables, assignment,operators, `if/else`) $\Rightarrow$ (5,5,3,2) would represent a student who has shown their ability is sufficient in variables and assignment, but whose knowledge isn't yet complete on operators or `if/else` statements. Updating the profile would be a function which takes as its inputs, the students profile *P,* the question they were asked *Q*, and answer they gave *A*. The output of this function would be the students new profile *P'*. A suitable question for this student would be one which examines further their ability in the latter areas, to accurately profile their ability.

An example of how this could be used would be when a student fails to answer a question on assignment statements correctly, their updated profile *P'* will acknowledge this by lowering their assumed knowledge for the area. Through profiling students in this manner, certain assertions can be made about what concepts students are familiar with, and the next concept to be learned can be dynamically decided based on each student's current knowledge. This will aid greatly in deciding the best order to teach programming concepts.

For the following example, a subset of the Java programming language will be used and the question is focused on basic program syntax and conditional statements. Assessment is the key to all learning (Richard, 1998), so naturally discovering how students learn is best done through assessment. Given that the answers provided by students will be subject to machine analysis, multiple choice questions seem the most suitable to examine students' ability. An example of a typical multiple choice question, that could make reasonable assumptions about students' abilities with simple operations and `if/else` statements is shown in figure 1. This example assumes the student has a profile similar to the one above, i.e. they have completed lessons in variables, assignments, operators and `if/else` statements, but seems to lack knowledge in the latter areas.

```
int x = 89%9; int y = 89/9; int max = 17;
if(x==8 && y>x)
```

```
{
if (x+y < max ) { System.out.print("Hello");}
else{System.out.print("Goodbye");}
}
```

Q. What is the output of this code?

a) No Output

b) `Hello`

c) `Goodbye`

d) `Hello Goodbye`

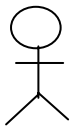
e) `Goodbye Hello`

Figure 1: An example question

In this example there are answers that clearly show if a student has extreme weakness in certain areas. A student who chooses option 'a' probably believes that the outer if statement evaluates to false, and therefore does not understand one or more of the %, /, ==, && or > operators in java. The student's profile should be adjusted to reflect this. An alternative interpretation of the student's response is that they do not understand conditionals at all; but their profile of '2' would be inconsistent with this. If a student answers 'b' it could be presumed that the student was slightly tricked by the question, they have not a full understanding of `if` statements or of the operators used. However as 'b' is a feasible answer, it cannot be assumed that they have no knowledge of the area. Students who answer 'd' or 'e' clearly have no understanding of `if/else` statements, as they requires both the `if` and the `if` to be executed. In particular, 'e' is extremely infeasible as it requires that the `if` be executed before the `else`. This analysis is summarised in the diagram seen in figure 2, together with the types of adjustment that would be made to the profile model.

It can be seen in the this example that meaningful probablistic results can be drawn from even a reasonably simple multiple choice question by assigning each answer a level of feasibility. Over a series of such questions, it would be possible to draw accurate conclusions about a student's programming ability, and where their programming difficulties lie. This information can then be used to determine the learning patterns of students. For example, it could determine if students understand recursion better if it is taught before or after iteration. It has been claimed that the order these concepts are taught in can affect students understanding of them (Turbak, 1999), a system such as this could offer some further insight. Furthermore, if a lecturer wished to examine new teaching techniques, e.g. using a new problem to demonstrate a programming concept or the use of a different analogy, they could have direct feedback regarding the impact of such techniques when the students next take their tests.

Student



P= (5,5,3,2)

Q= "What is the output…"

A= see below

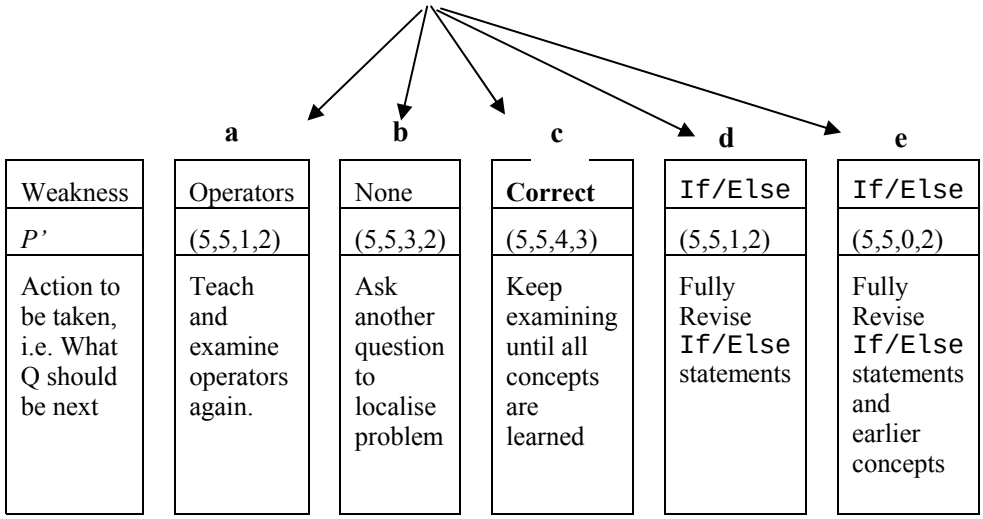


Figure 2: The adjustment of the profile

Design & Development Methodology

Shown in figure 3 is the development cycle that will be employed when building the proposed system. There are four major stages in each iteration, all of which are detailed below. This design methodology has been chosen because the lack of fundamental knowledge suggests that an evolutionary prototyping strategy(Su, 1997) has a better chance of success than one of the more traditional life cycle models. It is important to note that the tools that will test the tool, and determine its progress, must be included in development from the beginning to ensure that they can accurately measure its performance and furthermore, to ensure that the testing software itself improves during each increment.

Generating suitable questions

It is crucial that the supply of questions to students is of consistently high quality to ensure that the system can improve at each development increment. Accurately gauging students' abilities with respect to each concept will require questions suitable for each level of ability. If the system were to ask students' questions either greatly below or above the students current level, it would both de-motivate the students and mislead the profiler.

There are two possible ways of ensuring this does not happen, the first would be to have an *extremely* rich repository of suitable questions for the various concepts. Whilst this approach may seem feasible for a restricted subset, such as for the `if/else` example above, it is less likely to be feasible when the system is required to have a large number of suitable questions to cover all the fundamental concepts of programming. A second manner is currently being researched for question generation, that of employing genetic programming (Pillay, 2002) to generate sufficiently difficult pieces of code that the students can be examined on. This research is in its embryonic phase, currently only employing mutation to adjust the variables and levels of nesting, yielding programs that follow a similar syntax tree whilst looking reasonably different. For automatic generation of code that is challenging for the students to interpret, it would be desirable to create random paths through a syntax tree, merging different branches to yield vastly different programs focusing on similar concepts. This research must be furthered if genetic programming is to be used to generate examinable questions for students.

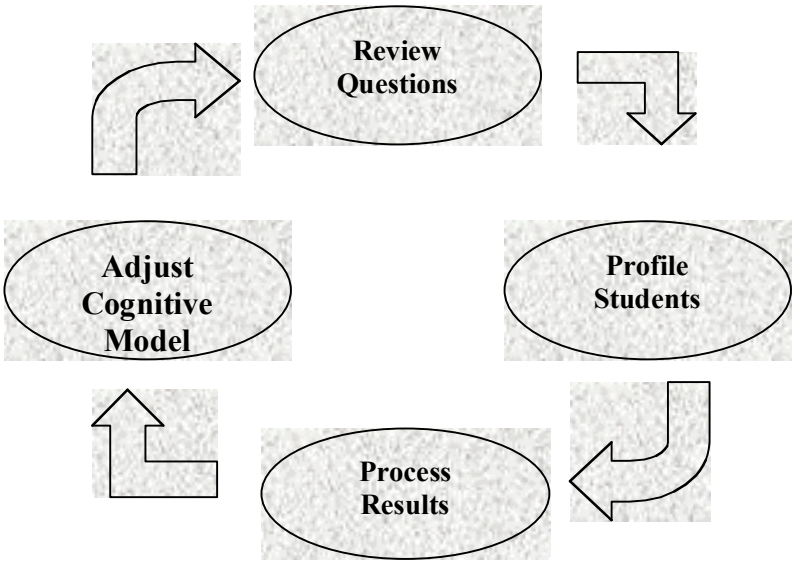


Figure 3: The development cycle

Developing the profiling system

The system that profiles the ability of the students will be subject to review at each iteration of development. It is clear that if a student answers a question correctly or incorrectly, their profile should adjust in some reasonable manner. It is crucial that the level of reaction from the profiler be chosen correctly to avoid over-correction, whilst still reacting suitably to infeasible answers. Through simple experimentation, we have already seen how difficult it is to define and implement the notinon of 'reasonable adjustment'. This is currently one of our main areas of research.

Processing results

Whilst students must learn from their mistakes, lecturers must learn from both their own and those of their students. If the profile produces results showing the students performed poorly in tests on a given subject, the lecturer must acknowledge that the topic has not been sufficiently covered, and that perhaps teaching techniques were inappropriate for the subject. Failure to acknowledge this would heavily restrict the ability of the system to improve itself. This type of direct feedback can aid the lecturer in discovering when and where the students have been left behind in certain topics. The lecturer must note which techniques worked well, and which ones clearly did not. In doing this, the lecturer will be creating a repository of teaching techniques that clearly benefit students, and this will aid us in building an understanding of how students learn. This type of knowledge is of huge importance for a lecturer and is referred to as pedagogical content knowledge(Gess-Newsome, 1999). The lecturer can also observe which ordering of concepts works better, which would give a good indication as to which are the best paths through the tree of learning programming language concepts, syntax and semantics.

Adjusting the cognitive model

Adjusting the cognitive model at each iteration can take place in two different manners. The first option is to adjust the parameters and weightings in our representation e.g. changing the perceived relative levels of difficulty of recursion and iteration. This would happen when the average scores for each concept differ greatly in an exam that involved both iteration and recursion.

The second option is to add new properties to the model, e.g. teaching declaration and assignment together to avoid confusion. A classic example of this is the observation that whilst some concepts may be taught separately, they seem to be learned in conjunction, i.e. students that understand one,

usually understand the other, as the concepts complement one another. This can be observed in exams where it is clear that students either grasp both concepts, or have little knowledge of either. An example of such a pairing would be the concept of recursive functions along with recursive data structures.

For this project it is assumed that both of the above are required in order to create a model capable of representing the complex issues within learning. The evolution of the cognitive model is essential for subsequent generations to ensure constant improvement of the system.

Implementation

Following from the cyclical design chosen, it is intended that the project be developed in a series of iterations, it's performance in an academic environment being measured at each stage. As the project is still in its youth, the details of the implementation are not yet concrete, however the following requirements will govern the implementation.

1. **Portability:** It is desired that the system be easily portable so as not to restrict the potential number of users. For this reason, Java will be chosen as the language of implementation.
2. **Duration of each Cycle:** Whilst the length of each cycle may fluctuate, it is preferred if the early cycles are reasonably short (preferably less than one month), so as to encourage rapid development. As the project approaches maturity the cycles will need to be longer to permit more exhaustive testing.
3. **Freedom of Software:** All ancillary software(e.g. database software, testing software etc.) must be freely available to the public. This is to ensure that the potential number of users is not restricted due to institutions being unwilling to pay for software merely to use the system.

The implementation of this project will consist of firstly building a working system that meets all the requirements and secondly forming a network of colleges and universities willing to participate in the project. The second goal is of huge importance as the evolution of the system depends heavily on a varied user base. It is desired that the working group will be both multi-institutional and multi-cultural, as this offers the advantages of an increased experience pool and a wider variety of student profiles. This will be extremely useful when attempting to generalise the results to form a reasonable cognitive model.

Applications & Future Uses

In the United States alone, the e-learning market is worth in excess of 10 billion dollars(Hall, 2002). Computers are now used to teach a wide variety of subjects ranging from corporate training to linguistics. However, there is one area in which e-learning has failed to make a high impact, that of teaching computer science. Undoubtedly there are huge advantages to e-learning in an academic environment, the tools and materials are permanently available, independent of both time and location. This outperforms even the most enthusiastic of lecturers. However, despite these advantages many modern attempts to teach computer science electronically seem to fail. Unlike corporate training where the material is reasonably simple to digest, or linguistics where there is a large body of literature concerned with its education, computer science education is neither simple nor well documented. Some of the more important subjects in education theory such as epistemology and pedagogy have been largely ignored thus far in computer science education and as a result electronic teaching methods suffer.

The full development of the proposed system should have significant implications for both traditional teaching methods and the new wave of learning technologies. Traditional teaching can benefit from the development of a cognitive model by adjusting the teaching methods to complement the natural learning methods of students. The benefits to e-learning are potentially huge; this system addresses many of the weaknesses exhibited by adaptive learning technologies, a fuller understanding of how

students learn would vastly increase the performance of these technologies, thus unlocking their true potential.

In conclusion, it is clear that for research in computer science education to progress a cognitive model is required. The proposed system will empirically develop a reasonable cognitive model that will be capable of answering many questions about how students learn.

References

Dunican, E Making the analogy: alternative delivery techniques for first year programming courses. *14th Workshop of Psychology of Programming Interest Group*, Brunel University, June 2002.

Gess-Newsome, J., & Lederman, N.G. (Eds.) (1999). *Examining Pedagogical Content Knowledge.* Kluwer Academic Publishers.

Adkins, S. S. (2001) *Market Analysis of the 2002-2010 U.S e-learning Industry: Convergence, Consolidation and Commoditization*. Sunnyvale, CA: Brandon-Hall.

Holmboe, C., McIver, L., and George, C. (2001) Research agenda for computer science education. *14th Workshop of Psychology of Programming Interest Group,* Bournemouth,.

McCracken, M. and Almstrum, V. and Diaz, D. and Guzdial, M. and Hagan, D. and Kolikant, Y.B.-D. and Laxer, C. and Thomas, L. and Utting, I. and Wilusz, T. (2001) A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM Bulletin Vol 33*.

Pillay, N. (2002) Using genetic programming for the induction of novice procedural programming solution algorithms. *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp 578-583.

Sprinthall, R. C., Sprinthall, N A., and Oja, S. N. (1998) *Educational Psychology*, 7th Edition, McGraw-Hill Education.

Stojmenovic, I. (2000) Recursive algorithms in computer science courses: Fibonacci numbers and binomial coefficients. *IEEE Transactions on Education*, Vol 43, No.3 , August 2000.

Su, S.Y.W. and Chatterjee, R. (1997) KBMS-based evolutionary prototyping of software systems. *8th IEEE International Workshop on Rapid System Prototyping*.

Turbak, F., Royden, C., Stephan, J. and Herbst, J. (1999) Teaching recursion before loops in CSI. *Journal of Computing in Small Colleges*, Volume 14, Number 4, May 1999, pp 86-101.

van Roy, P., Armstrong, J., Flatt, M. and Magnusson, B. (2003) The role of language paradigms in teaching programming, *SIGSCE* 2003 (Feb 19-23, 2003, Reno, NV.)