A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition

Jerry Mead
Computer Science Department
Bucknell University
mead@bucknell.edu

Simon Gray
Department of Mathematics and Computer Science
College of Wooster
sgray@wooster.edu

ABSTRACT

Traditional approaches to programming education, as exemplified by the typical CS1/CS2 course sequence, have not taken full advantage of the long record of cognitive and experimental studies on the development of programming skills. These studies indicate a need for a new curricular strategy for developing programming skills and indicate that a cognitive approach would be a promising starting point. Taking this approach, this working group seeks to accomplish two major goals:

- to identify a Dreyfus stage [1] for programming proficiency that students should attain by the time they complete an undergraduate computer science degree program and
- to identify a collection of programming skill sets that students should have when they complete an undergraduage computer science degree program and to characterize each skill set by a set of expected outcomes, thus associating each skill set with a measurable level of mastery.

1. BACKGROUND

As educators we often have the belief that if we carefully prepared and presented material, our students learned it and could demonstrate their mastery of it. However, a record of experimental studies on the development of programming skills reveals that, despite our best intentions and efforts, our students do not have the programming mastery expected of them. The problem is international in scale.

The ITiCSE 2001 working group, “A multi-national, multi-institutional study of assessment of programming skills of first-year CS students” [6], was motivated by an oft-expressed concern that students are not as proficient at programming as we expect them to be. This concern raised two questions for the group: what should students be able to do at the end of their first year of programming study, and how can this be assessed? The group identified the ability to solve problems as the goal for a first year computer science student and prepared an assessment instrument and applied it at several institutions. The group reached the unhappy conclusion that first year students do not have the problem-solving skills expected of them.

Building on this work, the ITiCSE 2004 working group “A multi-national study of reading and tracing skills in novice programmers” [5] asked “Why did the students not fare well as programming problem solvers?” To answer this question, the group created a pair of assessment instruments to evaluate a student’s ability to (i) read and understand a block of code (trace through it to predict what it would do) and (ii) to fill in a small piece of missing code so that the completed block fulfilled a clearly specified purpose. The group’s conclusion was that many students “have a fragile grasp of skills that are a prerequisite for problem-solving.” That is, their understanding of essential programming concepts and when/how to apply them was insufficient to be effective at the higher level skill of problem-solving.

In a paper to be presented at SIGCSE’06 [2], another multi-national, multi-institutional study describes the design work of graduating computer science students is analyzed. The paper concludes that the majority of the graduating students studied cannot design a software system.

These studies seem somewhat surprising until one reviews research on the way practitioners acquire skills. Dreyfus and Dreyfus [1], for example, describe a sequence of five stages of skill acquisition along a continuum from novice to expert. When applied to programming, each stage characterizes a set of programming and problem solving capabilities we can expect a practitioner *at that level* to do well. Studies of programmers and programming activities (see [11] and [12] for reviews of the literature) reinforce this view of gradual development. Not surprisingly, they tell us that novice and expert programmers employ quite different strategies for reading and writing code. Novice strategies clearly reflect their low level grasp of the syntax and semantics of language constructs and their fledgling problem-solving skills, while expert strategies reflect their ability to problem-solve at higher levels of abstraction based on internalized patterns built from years of experience.

There seems to be a disconnect between the traditional approach to programming education, exemplified by the typical CS1/CS2 course sequence, and what researchers have discovered about cognitive development and skill acquisition. This suggests a need for a renewed discussion on pedagogical strategy for developing programming skills, and studies suggest that a cognitive approach would be a promising starting point.

2. PROJECT OVERVIEW

This working group is motivated by a desire to develop better programming skills in students over the span of an undergraduate degree program. To achieve this goal, it is critical to understand the process by which programming skills are acquired and the rate at which the process can occur. Of course it is also important to be able to assess whether the instruction delivered is meeting its objectives. We see three intertwined problems.

Problem I: How do we define programming skill levels that students should attain as they progress through a computer science degree program?

Problem II: How do we determine if the students have indeed attained these levels?

Problem III: What is the most effective way to teach these skills so that the expected outcomes are met?

These problems suggest an exploration from three perspectives. The first perspective is cognitive and derives from models of cognitive development, such as the five stages of development described in Dreyfus and Dreyfus [1] and summarized in section 3. The second prespective is evaluative and sees assessment as a critical component of any attempt to improve programming education.The third perspective is educational and highlights the pedagogical aspects of programming skill development.

Assessment can play an interesting role in the exploration of the three problems motivating the working group. An assessment strategy is clearly the answer to Problem II, but its development can work hand-in-hand with the solutions to the other two problems as well. Addressing Problem I, if something like the five Dreyfus development stages are to be the starting point for identifying levels of programming capability, then more clearly defined capabilities must be identified for each stage. One way to specify these stages would be to associate objectives and outcomes for each of the stages. These objectives and outcomes clearly form the heart of an assessment strategy, somewhat like pre and post-conditions for a method. But more importantly, we believe identifying objectives and outcomes will drive the exploration of Problem III, i.e., pedagogy of developing student programming skills at these levels.

It is clear that Problems I and II must be resolved before Problem III can be fully addressed. It is also clear that the work required for Problem III is substantial and of a different sort than that required for the other two problems. This working group, then, will focus on Problems I and II and leave Problem III for a follow-on working group.

The Dreyfus skill levels are a motivating factor for the working group and it is a major goal to determine a Dreyfus level of programming skill that is appropriate for students completing an undergraduate computer science program. Recognizing, however, that a programmer's skills cannot be characterized by a single stage of development, in addressing Problem I and II the working group will focus on identifying programming skill sets that a graduating student should have and the level of mastery the student should reach for each of the sets.

3. PROJECT PLAN

There have been many studies of how novice and expert programmers work, but little on how a novice actually progresses to the expert level. One goal of the working group is to contribute to the solution by studying the trajectory from novice to expert with an aim to determine a Dreyfus stage for programming skills of computer science graduates. We summarize the Dreyfus stages here.

Novice: A novice learns basic facts, terminology, and rules and how to apply them in well-defined circumstances. In this sense, everything novices do is largely "context free;" they are not asked to reason about the rules.

Advanced Beginner: The advanced beginner starts to develop a feel for the rules through repeated practical application. The elements become "situational;" the student is beginning to understand the use of concepts/rules in situations that are similar to those from prior examples. This represents a gradual extension of the student's learning from the controlled environment of the "context free" examples to novel, but familiar applications ("situations"). A goal of this stage is to develop more generalized understandings of the rules and when to apply them, leading naturally to the next stage.

Competence: Being competent means having a deep enough understanding of the rules to know when they are applicable and how to apply them in novel situations. It also means knowing how to create a plan; to look for and identify the facts relevant to a situation so that a decision can be made about which rule(s) to apply. Thus this level is characterized by conscious problem-solving.

Proficiency: The performer at this level has a more refined and internalized sense of the rules so is able to do a better job of filtering and evaluating the facts in a situation. A developing intuition (involving pattern matching of relevant facts against generalized experiences) leads quickly to suggestions for action, then conscious analysis is used to evaluate the alternatives.

Expert: Through extensive experience and reflection, experts produce increasingly abstract representations and become better at mapping novel situations to these internalized patterns. Experts often find it difficult to express their decision-making process in words.

Another goal of the working group will be to determine a group of programming skill sets that students should acquire during their studies. The skill sets should be seen as stages the students attain during the time of their degree studies; or alternatively as a sequence of approximations to the final Dreyfus stage expected of graduates. Each skill set will be defined hand-in-hand with a set of goals and outcomes for the skills. In this way an assessment strategy will be a natural outcome of the groups efforts.

4. GOALS AND ACTIVITIES

Once the membership is established, the working group will begin its formal activities. The proposers will put together an initial reading list and distribute it. Based on feedback from the group, a final reading list will be distributed. Prior to convening in Bologna, members of the working group will:

- do readings from the distributed reading list;
- produce a first estimate of the Dreyfus stage that computer science graduates should attain;
- produce a first approximation for the programming skill sets that a computer science graduate should attain;
- produce a shared understanding of the terms "outcomes" and "objectives";
- solidify expected outcomes for the meeting in Bologna;
- prepare an agenda for the meeting.

The group's goals during ITiCSE 2006 are to:

- determine a Dreyfus stage that computer science program graduates should attain.
- identify a group of programming skill sets that characterize the stages of programmer development;
- begin to produce descriptions of objectives and measurable outcomes for the stages of programmer development;
- prepare the group's presentation to the conference.

The proposers understand that determining programming skill sets with accompanying measurable outcomes for the development stages is a challenging talk. However, the two activities naturally go hand-in-hand, and so will be more easily tackled in tandem.

After the conference concludes:

- a subgroup will take responsibility for writing a draft of the final report;
- those willing to continue will prepare a proposal for a follow-on working group to explore Problem III.

5. REFERENCES

[1] H. Dreyfus and S. Dreyfus. *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer.* Free Press, New York, NY, USA, 1986.

[2] A. Eckerdal, R. McCartney, J.E. Mostrom, M. Ratcliffe, and C. Zander. Can graduating students design software systems? *SIGCSE 2006 Proceedings, to appear*, 2006.

[3] Simon Gray. *Abstract Data Types and Object-Oriented Programming with Java.* Addison-Wesley, Boston, MA, USA, in press.

[4] Simon Gray, Stephen Edwards, Gary Lewandowski, and Anil Shende. Improving student programming skills by developing program comprehension abilities: panel discussion. *J. Comput. Small Coll.*, 20(3):235–237, 2005.

[5] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 119–150, New York, NY, USA, 2004. ACM Press.

[6] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In *ITiCSE-WGR '01: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 125–180, New York, NY, USA, 2001. ACM Press.

[7] Jerud J. Mead. Vmoe: A virtual machine operating environment for the computer systems curriculum. *Computer Science Education*, 1(2), 1989.

[8] Jerud J. Mead. A compiler tutorial scaled for the programming langauges course. *SIGCSE 2006 Proceedings, to appear*, 2006.

[9] Jerud J. Mead and Anil M. Shende. *Persuasive Programming.* Franklin, Beedle, & Associates, Wilsonville, OR, USA, 2001.

[10] Jerud J. Mead and Anil M. Shende. *A Guide to Persuasive Programming using Java.* Franklin, Beedle, & Associates, Wilsonville, OR, USA, 2002.

[11] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[12] Leon E. Winslow. Programming pedagogy – a psychological overview. *SIGCSE Bull.*, 28(3):17–22, 1996.