FIRST PRINCIPLES OF CS INSTRUCTION

Katrin Becker
Graduate Division of Educational Research,
University of Calgary
Calgary, Alberta, Canada
(403)932-6322
becker@minkhollow.ca

Abstract

Much attention has been paid in recent years to finding more flexible and less prescriptive approaches to the design of instruction than those put forward during the latter part of the twentieth century. A view of instruction as causal and largely behaviouristic has given way to one that is guided and primarily constructivist. In his current research, David Merrill outlines five fundamental principles of instruction which have broad implications for teaching computer science (CS). These five principles are: 1) solving real-world problems, 2) activating existing knowledge to build new knowledge, 3) demonstrating new knowledge to the learner, 4) allowing learners to apply new knowledge, and 5) integrating knowledge into the learner's world. The following paper describes these principles and discusses how they related to instruction in CS.

1.0 Introduction

M. David Merrill's career in instructional technology, with a career has spanned 40 years, and include numerous significant contributions to the field. He is probably best known for his Component Display Theory [1]. In the 1990's he was one of the foremost proponents of "Second Generation Instructional Design (ID2)" [2] which acknowledged a more open-ended and less prescriptive approach to ID, and included Instructional Transaction Theory[3] and ID based on knowledge objects.

The First Principles of Instruction are the result of a systematic review of instructional design theories, models and research. Each of the principles included satisfies the following properties: 1) promotes more effective, efficient or engaging learning, 2) is supported by research, 3) is general enough to apply to any delivery system or instructional methodology, and 4) is design oriented with direct relevance to promoting learning activities. Merrill defines a 'principle' as a basic method, and describes it as a "relationship that is always true under appropriate conditions regardless of program or practice (variable methods)." [4, p43] There are five principles that constitute a set of fundamental elements common to all effective instructional design. Merrill hypothesizes that 1) "Learning from a given instructional program will be facilitated in direct proportion to the implementation of first principles of instruction", and 2) the "learning from a given instructional program will be facilitated in direct proportion to the degree that first principles of instruction are explicitly implemented rather than haphazardly implemented." [4, p43] If true, then we stand to gain from examining how Merrill's First Principles apply to computer science instruction, which is what the following paragraphs will do.

2.0 Computer Science Education

Computer Science is fortunate to have two international organizations (ACM & IEEE) who together have produced widely accepted curricula describing undergraduate programs in computing [5]. Although it is and will probably remain a subject of debate, they nonetheless stand as standards against which programs and courses can be compared. These extensive documents go a long way towards describing *what* computer science is, but even though some example courses are suggested the purview of this document does not really extend to theories or models of *how* to teach computer science.

Instructional Design methods for teaching science and math receive considerable attention [6, 7], but less attention is paid specifically to computer science education, and a search at *GoogleScholar* for "instructional design" and "computer science education" turns up fewer than 300 hits which drops to < 140 if we look at only the last ten years. The vast majority of these references concern themselves with web-based and distance education, and the use of knowledge objects. Specific ID theories and models are rarely mentioned in the context of designing instruction for computer science education.

Several areas in computer science have had significant influence over learning and instructional theories, notably human-computer interfaces, and artificial intelligence, but it does not seem to be a reciprocal relationship. On the whole, learning and ID theories have had relatively little impact on how computer science is taught, and a great many courses are still taught much the same way they were a generation ago. Admittedly, we have always made extensive use of technology, which has evolved over time, but in many cases the technology is used less as a system for the support of learning and more as an apparatus on which to run programs and develop documentation.

3.0 Merrill's First Principles of Instruction

According to Merrill, the success of a given instructional program will be directly proportional to how well and how deliberately the first principles are implemented. [4] First principles are intended to be domain NON-specific and so should be applicable to almost any discipline. The following paragraphs provide a general description of these first principles, and the subsequent section relates each to computer science education.

3.1 Problem

"Learning is facilitated when learners are engaged in solving real-world problems." Problems should be authentic, real world, and personal. Not mentioned by Merrill, the possibility for fantasy to play a role should also be considered as these can have authentic and personal elements that result in highly motivating scenarios. Learning to solve problems involves several levels of instruction, from understanding the problem through identifying the tasks and implementing the operations that make up the tasks. Often instruction concentrates on the implementation level and fails to involve learners in the problem level. Complex problems can be mastered only with adequate coaching, or after sufficient time has been spent building up from simpler problems.

3.2 Activation

"Learning is facilitated when existing knowledge is activated as a foundation for new knowledge." Also described as scaffolding – this involves making explicit connections between new knowledge and past experience. This must be more than

merely testing prerequisite knowledge. It involves activating those mental models that are relevant to the current context – with an emphasis on *active* participation.

3.3 Demonstration

"Learning is facilitated when new knowledge is demonstrated to the learner." This principle tells us we must *show* people what we want them to learn, not simply tell them. Included here are examples as well as counter examples, demonstrations, visualizations, and modeling. Media should play an instructional role, multiple representations should be used, and learners should receive appropriate guidance for such things as: finding relevant information and explicit comparison of demonstrations.

3.4 Application

"Learning is facilitated when new knowledge is applied by the learner." Learners must be guided in their problem solving. This involves a coaching process where more guidance is offered to start, including error detection and correction, gradually withdrawing assistance until they are able to manage on their own. Merrill says, "Appropriate practice is the single most neglected aspect of effective instruction." And as we all know, "You can't rush Mother Nature." Unfortunately, education cannot be made into an exercise in efficiency. Learning is a natural process and takes *time*.

3.5 Integration

"Learning is facilitated when new knowledge is integrated into the learner's world." In and of themselves, animation, multimedia, even games have only a temporary effect on motivation. Ultimately, learning is its own reward. A case in point is the video game industry, where meeting challenges and demonstrating improvement are the motivators that have made this industry into a multi-billion dollar business. In games, once a new skill is learned, players seek ways to apply that skill and improve upon it. The popularity of games is less about winning than it is about meeting challenges and showing improvement. This is also true in real life. Reflection, discussion, and finding new ways to use new knowledge and skills all facilitate learning. Students quickly learn to recall and recite on cue, but the information they so readily regurgitate is not available to them for application elsewhere. Knowledge that is not used remains inert.

4.0 First Principles Applied to CS

This section applies the principles defined above to computer science instruction, using examples. Note that each principle interconnects with others, so that a single, well-chosen example can meet the criteria for all of them. The more of these carefully-chosen problems we have in our courses, the more effective the instruction will be.

4.1 Problem (engagement in solving real-world problems)

Some would say that problem solving is the very core of all computer science, and one would be hard-pressed to find a programming class of any description that did not include assignments posed as problems to be solved. However the kind of problem-based learning proposed as Merrill's first principle is more deliberate and structured [8-10] than most typical CS assignments. This kind of problem-based learning is distinct from showing examples, and also from assigning a problem and leaving them to it. It involves a structured, *guided* approach that begins with a delineation of the problem and its domain, and progresses through the research and discovery of relevant knowledge and data, and finally through a presentation of a solution and a reflection on what was learned

and deliberate and conscious connection with existing knowledge.

Problems should be relevant – real-life, but also problems that consider the characteristics of the learners themselves. We want the problems to be interesting for our students while still forcing them into contact with the necessary content. If building on a student's expertise results in a more engaging problem, we should look seriously at how students spend their time. Problems drawn from accounting or management are common in CS, but seriously, how many students do you know that do accounting as a hobby or pastime? They do, however, play games. For many students, digital games provide a rich source of problems that they will find both challenging and engaging. [11] In more senior courses it is often possible to allow students themselves to propose problems to solve, but in introductory classes, they will typically require more direction. Still, it is possible to offer a choice among several alternatives, where each problem to be solved addresses the same conceptual issues. For example, the classic arcade game *Frogger*, a hospital emergency room simulation, and a dynamic restaurant menu system are all problems that lend themselves to the use of inheritance and polymorphism. If the pedagogical objectives are clearly defined, then the subject matter can become flexible, thereby creating relevance, and the potential for a personal connection with the problem.

4.2 Activation (activate existing knowledge as foundation for new knowledge)

To activate existing knowledge, we must know what that existing knowledge is. These days, most probably have cell phones, know how to do email, surf the web, find music and videos, and play games. They will feel themselves quite proficient, yet in reality will typically lack sophistication in such things as searching and the critical assessment of resources. It is important to acknowledge their skills, without either assuming too high a level of sophistication, nor too much nescience. One will discourage students, while the other will bore them. Either way, they will become disengaged and motivation will suffer.

Introducing the notion of greedy algorithms by examining or developing an algorithm for making change is one example of activating existing knowledge. These days, few people seem to calculate change themselves or count it out from the amount charged, but the problem of how to allot a minimal number of bills and coins still has a real-life connection that can be used to build an effective bridge. Another example could use the typical human approach to searching for a name in a phonebook as a means of introducing a binary search. As an aside, this particular example will probably lose its relevance soon, as searching becomes more and more automated. It will then become necessary to locate other examples – perhaps finding a specific music CD in a collection, or one's final grade in a large class list. This highlights a key point: relevance changes over time. If we wish to start from where the students are, then we must be prepared to assess the knowledge they bring to the situation regularly.

4.3 Demonstration

Learning objects can be useful here, although as with any other resource, they should be carefully assessed before they are used. [12] Students report that live, in class demonstrations of programs are more effective than simply reading through and explaining code. Another effective mechanism is modeling behavior such as answering a question by performing a quick search on the internet, which of course would include a

demonstration of rapid assessment of potential resources. Instructor competence with the technologies available is crucial if students are to be expected to demonstrate competence themselves. Credibility is lost if the instructor cannot give the impression that he or she is able to solve the problem using the tools available to the students.

Another approach often reported as welcome by students is to develop a solution to a problem on the fly – including errors and blind alleys. One dilemma for many novices is that the process used by the instructor in class when developing a solution to a problem is one that has typically been well-rehearsed and delivered many times, yet when the novice attempts to emulate this approach later, the result is not nearly so wrinkle-free. We often learn a great deal from our mistakes and to some extent we can also learn from the mistakes of others. We should allow our students to observe error recovery.

4.4 Application (by the learner)

The computer science curriculum is overflowing with content. The body of knowledge associated with the discipline has grown and evolved over the last 40 years and in our desire to provide students with as much information and knowledge as we can, we sometimes forget that they still need time to absorb the information - and this includes time for practice. Chess masters, musicians, swimmers, and others have been shown to require on average ten years to achieve expert status! [13] What makes us think we can create expert programmers and computer scientists in just four?

We can no longer cover the same ground in an undergraduate program as we did when we were students, yet we cling to a desire to do so, and to add all that we have learned since graduating as well. Modern theories of education, including Merrill's imply that students will be better prepared through the acquisition of deep knowledge in fewer areas than through a shallow or cursory acquaintance with many. We do our students a disservice by attempting to move on to the next topic too quickly, and often in our eagerness we end up teaching content that is "a mile wide but only an inch deep". One the other hand some topics can be taught using a spiral approach. The topic can be introduced but treated superficially in one course, and then addressed in greater depth in one or more subsequent courses. Recursion, algorithm analysis, and program testing are all topics that lend themselves to such an approach. Others, like ethics, professional practice, and communication skills can also be broken up and spread across multiple courses, but lend themselves more to sectioned, in-depth study than to layering from superficial overviews to deep learning.

4.5 Integration (new knowledge is integrated into the learner's world)

It is like coming full circle – we start from where the student is, and end by helping them convert new knowledge into a new starting point. If the curriculum is well integrated, subsequent courses can quite literally pick up where the others left off. If it is not, time will be required in each course to re-assess what knowledge the students bring.

The assignment component of the freshman class taught by the author on and off for over twenty years consisted of no less than six nor more than ten programming assignments – each one focusing on a different concept or set of concepts. The time pressure is high, and students barely get one handed in than they are headed full speed into the next one. There is no time in between to reflect, and in an effort to maintain

interest, each problem bears little relationship to the last. This kind of approach does not promote integration of learning. While it is sometimes impractical to create assignments that logically follow one another, attempts should be made to do so. When this is not possible, both practical and conceptual connections can be made explicit.

At the course level, content can and should be tied to the students' lives, current events, research, and professional practice whenever possible. For example, reading an article like Walpole's "Designing Games for the Wage Slave" [14] connects well with student's lives, can be used to discuss software design on many levels, *and* can be used as the basis for assessment criteria on a programming assignment.

5.0 Conclusions

Most courses can benefit from a more deliberate approach to the design of instruction, and following the principles laid out here can help. Freshmen have a broader base of experience than students had even five years ago, and integrating this experience into our courses forms connections that engages students in ways not otherwise possible. For example, today's freshmen have never really known a world without PCs, the internet and computer games. Most own cell phones and some sort of high-capacity digital music player that doubles as general mass storage. These are as much a part of their immediate surroundings as telephone land lines and televisions were for the generation before. Contextualizing their experience draws them in, and maintaining relevance while guiding their practice will sustain them. In short, following these first principles will not only help students learn *about* computer science, but will also teach them how to *be* computer scientists.

6. References

1. Merrill, M.D., Component Display Theory, in Instructional-design theories and models, C.M. Reigeluth, Editor. 1999, Erlbaum: Hillsdale, N.J. p. 279-333.
2. Merrill, M.D., Z. Li, and M.K. Jones, Second generation instructional design. Educational Technology, 1991. 30(2): p. 7 - 14.
3. Merrill, M.D., Z. Li, and M.K. Jones, Instructional transaction theory: an introduction. Educational Technology, 1991. 31(6): p. 7 - 12.
4. Merrill, M.D., First Principles of Instruction. Educational technology research and development: ETR & D, 2002. 50 Part 3: p. 43-60.
5. Computing Curricula 2001: Final Report of the Joint ACM/IEEE-CS Task Force on Computer Science Education, E. Roberts and G. Engel, Editors. 2001, IEEE Computer Press: Los Alamitos, CA.
6. Mintzes, J.J., J.H. Wandersee, and J.D. Novak, Teaching science for understanding: a human constructivist view. 1998, San Diego: Academic Press. xx, 360.
7. Taylor, P., P.J. Gilmer, and K.G. Tobin, Transforming undergraduate science teaching: social constructivist perspectives. 2002, New York: Peter Lang. xxiv, 481 p.
8. Jonassen, D.H., Learning to solve problems: an instructional design guide. Instructional technology & training series. 2004, San Francisco, CA: Pfeiffer. xxiv, 218 p.
9. Schank, R.C., T.R. Berman, and K.A. Macpherson, Learning by Doing, in Instructional-design theories and models: vol. 2, a new paradigm of instructional theory. 1999, Lawrence Erlbaum Associates: Mahwah, N.J. p. 161-181.
10. Savery, J.R. and T.M. Duffy, Problem based learning: An instructional model and its constructivist framework, in Constructivist Learning Environments: Case Studies in

Instructional Design, B.G. Wilson, Editor. 1996, Educational Technology Publications: Englewood Cliffs, NJ. p. 135-148.
11. Becker, K. and J.R. Parker. All I Ever Needed to Know About Programming, I Learned From Re-writing Classic Arcade Games. in Future Play, The International Conference on the Future of Game Design and Technology. 2005. Michigan State University, East Lansing, Michigan.
12. Wiley, D.A., The instructional use of learning objects. 2002, Bloomington, Ind.: Agency for Instructional Technology: Association for Educational Communications & Technology.
13. Ericsson, K.A., R.T. Krampe, and C. Tesch-Romer, The Role of Deliberate Practice in the Acquisition of Expert Performance. Psychological review, 1993. 100(3): p. 363-406.
14. Walpole, S., Designing Games for the Wage Slave. 2004, gamedev.net.