BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish excerpts from selected posts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/1666420.1666425 http://cacm.acm.org/blogs/blog-cacm

Too Much Programming Too Soon?

Mark Guzdial and Judy Robertson discuss the role of programming in introductory computer science.

From Mark Guzdial's "How We Teach Introductory Computer Science is Wrong"
http://cacm.acm.org/blogs/blog-cacm/45725

I've been interested in John Sweller and *Cognitive Load Theory* since reading Ray Lister's 2008 Australasian Computing Education Conference keynote paper, "After the Gold Rush: Toward Sustainable Scholarship in Computing." I assigned several papers on the topic to my educational technology class. Those papers have been influencing my thinking about how we teach computing.

In general, we teach computing by asking students to engage in the activity of professionals in the field: by programming. We lecture to them and have them study texts, of course, but most of the learning is expected to occur through the practice of programming. We teach programming by having students program.

The original 1985 Sweller and Cooper paper on worked examples had five studies with similar setups. There are two groups of students, each of which is shown two worked-out algebra problems. Our experimental group then gets eight more algebra problems, completely worked out. Our control group solves those eight problems. As you might imagine, the control group takes *five times* as long to complete the eight problems than the experiment group takes to simply read them. Both groups then get new problems to solve. *The experimental group solves the problems in half the time and with fewer errors than the control group. Not* problem-solving leads to better problem-solving skills than those doing problem-solving. That's when educational psychologists began to question the idea that we should best teach problem-solving by having students solve problems.

The paper by Kirschner, Sweller, and Clark (KSC) is the most outspoken and most interesting of the papers in this thread of research. Their title states their basic premise: "Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching." What exactly is minimal instruction? And are they really describing *us*? I think this quote describes how we work in computing education pretty well:

"There seem to be two main assumptions underlying instructional programs using minimal guidance. First[,] they challenge students to solve "authentic" problems or acquire complex knowledge in information-rich settings based on the assumption that having learners construct their own solutions leads to the most effective learning experience. Second, they appear to assume that knowledge can best be acquired through experience based on the procedures of the discipline (i.e., seeing the pedagogic content of the learning experience as identical to the methods and processes or epistemology of the discipline being studied; Kirschner, 1992)."

That seems to reflect our practice, paraphrased as "people should learn to program by constructing a program from the basic information on the language, and they should do it in the same way that experts do it." The paper then presents all the evidence showing that this "minimally-guided instruction" does not work:

"After a half-century of advocacy associated with instruction using minimal guidance, it appears that there is no body of research supporting the technique. In so far as there is any evidence from controlled studies, it almost uniformly supports direct, strong instructional guidance rather than constructivist-based minimal guidance during the instruction of novice to intermediate learners."

There have been rebuttals to this article. What's striking about these rebuttals is that they basically say, "But not problem-based and inquiry-based learning! Those are actually guided, scaffolded forms of instruction." What's striking is that *no one challenges KSC on the basic premise, that putting introductory students in the position of discovering information for themselves is a bad idea*! In general, the educational psychology community (from the papers I've read) says that expecting students to program as a way of learning programming is an ineffective way to teach.

What should we do instead? That's a big, open question. Pete Pirolli and Mimi Recker have explored the methods of worked examples and cognitive load theory in programming, and found that they work pretty well. Lots of options are being explored in this literature, from using tools like intelligent tutors to focusing on program "completion" problems (van Merriënboer and Krammer in 1987 got great results using completion rather than program generation).

This literature is *not* saying *never* program. Rather, it's a bad way to *start*. Students need the opportunity to gain knowledge first, before programming, just as with reading. Later, there is an *expertise reversal effect*, where the worked example effect disappears, then *reverses*. Intermediate students do learn better with real programming, real problem-solving. There *is* a place for minimally guided student activity, including programming. It's just not at the beginning.

Overall, I find this literature unintuitive. It seems obvious to me that the way to learn to program is by programming. It seems obvious to me that real programming can be motivating. But KSC respond to this, too, noting that "it is easy to share the puzzlement of Handelsman et al. (2004), who, when discussing science education, asked":

"Why do outstanding scientists who demand rigorous proof for scientific assertions in their research continue to use and, indeed defend on the bias of intuition alone, teaching methods that are not the most effective?"

This literature doesn't offer a lot of obvious answers for how to do computing education better. It does, however, provide strong evidence that what we're doing is *wrong*, and offers pointers to how *other* disciplines have done it *better*. It's a challenge to us to question our practice.

From Judy Robertson's "Introductory Computer Science Lessons—Take Heart!"

http://cacm.acm.org/blogs/blog-cacm/46781

I was somewhat alarmed to read Mark Guzdial's excellent and thought-provoking post, which argues that the way we teach introductory computer science is wrong. His argument is that some of the educational psychology literature claims that minimally guided instruction techniques, such as discovery learning, constructivism, and problem-based learning, are less effective than strongly guided instruction techniques. As an extension to this: teaching programming through the practice of programming itself is not effective for novices. As a lecturer of a first-year programming module myself, I spluttered into my cup of tea and hurried off to read the Kirschner, Sweller, and Clark article Mark recommended.

Kirschner, Sweller, and Clark have some strong words to say against minimally guided instruction approaches. For example, "The goal of instruction is rarely simply to search for or discover information. The goal is to give learners explicit guidance about how to cognitively manipulate information in ways that are consistent with a learning goal, and store the result in long-term memory." But hang on: in higher education we generally regard it as important that students know how to search and discover information for themselves. They require skills in self-directed learning. In the context of programming, for example, we may wish them to know how to look up documentation. We would also generally expect them to be able to search for information sources in the first stage of carrying out a research project. I suspect this is a question of the stage of cognitive and metacognitive development the learner is at in first year, and whether it is reasonable to expect more of them than manipulating information and storing it in long-term memory.

The authors also write: "[It] may be a fundamental error to assume that the pedagogic content of the learning experience is identical to the methods and processes (i.e., the epistemology) of the discipline being studied and a mistake to assume that instruction should exclusively focus on methods and processes."

I don't think that introductory computer science teaching *does* focus only on methods and processes. In fact, it is a bit of a straw man to consider what goes on in first-year computer science classes as pure minimally guided instruction anyway. Obviously there is a huge range of teaching approaches to novice programming across the world, but let's take Barnes and Kölling's *Objects First With Java* textbook and the BlueJ environment. It's very popular and used as an introductory text in many computer science departments. One of the features of this well-designed textbook is that it aims to teach high-level concepts as a priority over lower-level language constructs. The BlueJ environment enables students to experiment with object orientation by calling methods on objects in a graphical environment. The textbook encourages students to read code before they write it, and "wire in" small segments of their own code into a prewritten program. The lecture slides that come with the book give specific instruction and worked examples; students typically receive this sort of instruction before working on small examples in the lab. In fact, working on small examples after a lecture on programming concepts is, in my experience, a fairly common pattern in first-year instruction.

Kirschner, Sweller, and Clark recommend: a) providing worked examples for students to read and, b) providing process worksheets that explain to students the processes they should go through when solving problems. These are sensible suggestions, but I wouldn't say they are unusual for computer science teaching. I would suggest that we tend to use a mixed bag of instructional techniques rather than basing our pedagogy on pure theory. And therefore, we probably get our first-year teaching right at least part of the time. Which is a bit of a comfort.

Mark Guzdial is a professor at the Georgia Institute of Technology. **Judy Robertson** is a senior lecturer at Heriot-Watt University.