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 selected posts or excerpts.

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

DOI:10.1145/2754947 http://cacm.acm.org/blogs/blog-cacm

Bringing Evidence-Based Education to CS

Mark Guzdial says computer science teachers should use more evidence, less intuition.

Mark Guzdial
Computing Education Must Go Beyond Intuition: The Need for Evidence-Based Practice
http://bit.ly/1DdxT3a
February 22, 2015

The highest-quality education practice is *evidence-based education*, defined by the U.S. Department of Education (http://1.usa.gov/1CWNB4o) as "the integration of professional wisdom with the best available empirical evidence in making decisions about how to deliver instruction." The movement toward evidence-based practice has swept through medicine (http://1.usa.gov/1IttwlY), psychology (http://bit.ly/1CBcRM2), and other fields.

Computing educators' practice would dramatically improve if we drew on evidence rather than intuition. Raymond Lister has been writing for years about the problem that computing education practitioners do not engage with evidence and the research on education (http://bit.ly/1EMqBY1). Lister calls the knowledge computing educators use in making teaching decisions "folk pedagogy" (http://bit.ly/1NL9xmq).

Folk pedagogy encourages what is believed to be best practice, but cannot validate best practice.

We have evidence computing teachers do not use evidence. Davide Fossati and I studied 14 CS teachers from three institutions (http://bit.ly/1BV9uNo). Fossati asked them about times they made a change in their teaching practice; why did they make the change, and how did they know if it was successful or not. They used intuition, informal discussion with students, and anecdotes. *Not a single teacher* used evidence such as class performance on a test or homework.

Without evidence, teachers rely on intuition informed by experience. Sometimes that intuition may be informed by years of experience. Sometimes that experience is not at all relevant.

The March 2015 issue of *Inroads* (http://bit.ly/1ItulLy) has a section on "The role of programming in a non-major, CS course." That is a topic of great interest to me, since I have been developing and studying a programming-based approach to teaching introductory computing we have used with non-CS majors at Georgia Tech for over 10 years (http://bit.ly/1AkpH2x). The article (http://bit.ly/1NLcoff) by Richard Kick and Frances Trees describes the AP CS Principles course, its research-based development, and experience at the pilot sites. The article by Steve Cooper and Wanda Dann (http://bit.ly/19ND6ok) presents findings from early research on teaching programming to children, and evidence from courses at Stanford and CMU using Alice. The other papers use no evidence.

Henry Walker's article (http://bit.ly/1EMkm6s) argues programming should not be a priority for a non-CS majors course:

Development of student problem-solving skills (with programming) requires time, so the inclusion of programming within a non-CS majors CS course comes at a price: significant other topics must be dropped.

The claim "significant other topics must be dropped" is empirical. Since there are many non-CS majors CS courses with programming described in the ACM literature, one could consider those courses and identify the significant topics dropped in favor of programming.

Henry does not consider these contrasting cases, but offers recommendations of what a non-major student *ought* to know about CS. Are CS faculty the right ones to define the learning goals for students who *do not* plan a career in computer science? When we designed Media Computation, we formed an advisory board of non-CS faculty to tell us what their students needed to know about computing. For example, a professor in architecture wanted his students to understand the difference between Photoshop and a CAD tool for manipulating a diagram. While both can "extend a line," Photoshop manipulates pixels, while CAD tools manipulate a vector representation that can be used to control automated devices like milling machines. I had not realized that was an important data representation learning objective for architecture.

Michael Goldweber makes two arguments in his paper against including programming in a non-CS majors course (http://bit.ly/1P3od1X). His first is like Walker's, in that he offers what *should* be in a non-CS majors course. He suggests an "*embarrassment* model of course and curriculum development" in which "one enumerates the topics for which, if their students did not know about/have experience with, one should feel embarrassment":

I assert this topics list does not include programming, not because programming is without merit, but because the inclusion of programming requires a time commitment that leaves an insufficient quantity of time for the key "embarrassment" topics. These include algorithmic problem solving, the use of abstraction to tame complexity, and the limits of computation. It is my position that it would be an embarrassment if a student coming out of a non-CS majors course could successfully program a solution to Selection Sort or write a program that draws a fractal tree, but have no idea of how to apply algorithmic problem-solving to real-world problems.

Goldweber suggests a real-world problem he believes non-CS majors should be able to solve after an introductory course to CS:

Given a graph representing cities and connecting highways, some of the cities house a Red Cross warehouse while one other city experiences a disaster; describe an algorithm for locating the closest Red Cross warehouse.

My suspicion based on research evidence is few CS majors or non-CS majors would be able to solve this problem, even after several classes (with or without programming). Transfer of knowledge is difficult to achieve, and students are particularly challenged to recognize a real-world problem is related to other knowledge they have learned (http://bit.ly/1BVbKnW). In the decades of studies that have tried to find such transfer, the research evidence is that computing courses do not help students develop general problem-solving skills (http://bit.ly/19NPK6O). That is just my suspicion. We could gather evidence to see if students get closer to achieving Goldweber's goal with a programming course or a non-programming course. He does not offer any evidence.

Goldweber's second argument is about the cost of programming in time and tedium. He dismisses the use of languages such as "Pascal, Java, Python" in which students engage in "wrestling matches with compilers over a missing or misplaced semicolon or squirrely bracket." He disparages graphical programming languages ("Scratch, Alice, Kodu") as "children's introduction to programming":

Should the learning outcomes for a child's introduction to the field be what we want for one's singular university-level non-major's course?

Betsy DiSalvo published a paper on a study of African-American teens who learned both Python and Alice (http://bit.ly/19MBo6u). There was not a clear winner. DiSalvo found the preference depended on the *careers* these students desired. Those interested in computer science as a career preferred Python. Those interested in media and design preferred Alice because they liked what they created, but also because the graphical nature made the high-level structure more evident. What one participant told DiSalvo sounds much like what Goldweber and Walker want students to achieve:

I like the top-down design (in Alice projects). We are able to break down the bigger problems into smaller problems and then even smaller problems, so it's a simpler way to file through. I think you could probably apply top-down design to anything in life.

The biggest concern I have for making education decisions without evidence is *who* is making the educational decisions *for whom*. Most CS faculty (reflected in the authorship of these articles) are white and male. Because we are CS faculty, most of our experience is rooted in being students in STEM fields.

Our intuition is likely wrong about non-CS majors. Most non-CS majors are not students in STEM. Non-CS majors are more diverse. We do not know their experience, goals, or desired careers. We should not assume we know how to teach them. We should not assume what non-CS majors need to know. When we want to know what CS majors will need in their careers, we ask industry advisors (as the CS2013 curriculum process did, http://bit.ly/1yHg0Gc). We know much less what non-CS majors might need to know in their careers. We need to gather evidence to determine what non-CS majors need to know about computing.

We need the humility to recognize what we do not know, and we need evidence to inform our decisions. You do not want your surgeon to apply the best practices of folk medicine on you. Our students deserve the best educational practices informed by evidence, not folk pedagogy informed by intuition.

Comments

Mark, I agree completely with your main points: CS educators should do more to base our practice on evidence rather than intuition, because that intuition is likely wrong. I would go further in several ways.

First, our intuition is likely wrong about CS and STEM majors too, even if we think we know what they need to know, since most of our students are unlike us.

Second, we should leverage evidence-based practices that give us more direct evidence about our students as they learn. For example, approaches like process-oriented guided inquiry learning (POGIL) and peer instruction help us observe students as they grapple with new ideas, so we can respond more quickly to problems. When my students work through a POGIL activity, I see where they need help, I can respond immediately, and I know what to revise for the future. It is easy for lecturers and their students to overestimate what has been "learned" until it has to be applied.

On the other hand, I am not sure how far behind we are in CS education; folk pedagogy still seems to be the norm on many campuses.

—Clif Kussmaul

Completely agree, Clif! I'm a huge fan of Peer Instruction (http://bit.ly/1y47mae) because it gives me evidence about my class, what they know and what they do not.

Carl Wieman reports over 70% of physics teachers are familiar with Physics Education Research results and use at least one finding in their teaching. We in CS Ed are not there yet.

—Mark Guzdial

Walker and Goldweber have written a response to this post on the ACM Inroads *blog, at http://tinyurl.com/mwqspgy.*

—Lisa Kaczmarczyk

Mark Guzdial is a professor at the Georgia Institute of Technology.