DOI:10.1145/3000612

Mark Guzdial and Briana Morrison

Education

Growing Computer Science Education Into a STEM Education Discipline

Seeking to make computing education as available as mathematics or science education.

COMPUTING EDUCATION IS changing. At this year's CRA Snowbird Conference, there was a plenary talk and three breakout sessions dedicated to CS education and enrollments. In one of the breakout sessions, Tracy Camp showed that much of the growth in CS classes is coming from non-CS majors, who have different goals and needs for computing education than CS majors.[a] U.S. President Obama in January 2016 announced the CS for All initiative with a goal of making computing education available to all students.[b]

Last year, the U.S. Congress passed the STEM Education Act of 2015, which officially made computer science part of STEM (science, technology, engineering, and mathematics). The federal government offers incentives to grow participation in STEM, such as scholarships to STEM students and to prepare STEM teachers. Declaring CS part of STEM is an important step toward making computing education as available as mathematics or science education.

The declaration is just a first step. Mathematics and science classes are common in schools today. Growing computing education so it is just as common requires recognition that education in computer science is different in important ways from education in STEM. We have to learn to manage those differences.

High school students and teachers engaging in collaborative meetings about computer science represents an important step toward making computing education as available as science or mathematics education.

PHOTO BY CALVIN LIN, UNIVERSITY OF TEXAS, AUSTIN

Computer Science Is Invisible, Formally and Informally

Students enter mathematics or science classes at the post-secondary level with years of knowledge and experience behind them. Informally, many students talk to their parents about science issues ("Why is the sky blue?"), think about the numbers in their lives ("How much is the tax?"), visit science museums, and see media about mathematics and science. Students live in a world where living, chemical, and physical behavior is explained by biology, chemistry, and physics. They develop ideas about how the world works, some of which are wrong (like simple Lamarckian evolu-

a http://cra.org/wp-content/uploads/2016/07/BoomCamp.pdf
b https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

tion). Students start formal learning about mathematics and science in the earliest grades. Mathematics and science classes can help answer their questions and improve the theories that a reflective child has about the world.

While computing is ubiquitous in the developed world, and cellphones and other handheld computing devices are increasingly common in the developing world, few students get the opportunity to look under the covers of those devices to reflect on questions about computing. Maybe children might ask, "Why is my browser so slow?" The concepts that computer science explains are mostly invisible to children, such as digital representations, algorithms, and networks. If they don't see the computation in their world, it's difficult to reflect on and develop questions about it. We have less museum or media coverage than the rest of STEM. Few students enter secondary or post-secondary computer science classes with any previous formal education in computing.

The difference means it is difficult for a teacher to set expectations. Some students do have experience coming in to class; most do not. When does a student need remedial help? We are in a transitional stage where the gap between the haves and have-nots in computing education is large.

There is a positive side to the invisibility of computation in children's daily life. Mathematics and science students develop their misunderstandings of the world from daily life, too. Students' incorrect science theories are wrong, but mostly work. The world is not flat, but it seems so, and you can live much of your life believing the world is flat. On the other hand, students develop incorrect theories in computer science only in computer science classes and mostly because of how we taught them. If we change the way we teach, we can reduce misunderstandings.

Developing Learning Progressions

We have been doing mathematics and science education for a long time. We have a good idea of what students can do in science and mathematics from early ages, and how quickly they can develop new concepts and skills. Educators call this a learning progression.

Since computing education is younger than science and mathematics education, we do not have learning progressions. Computer science educators mostly have had the goal of preparing future professional programmers, but goals change when we talk about teaching everyone. Not every introductory biology student will become a biologist and not every intro physics student will become a physicist. We need to determine what pieces of computing the educated citizen needs to know. Then we can plan how and in what order we teach those pieces.

What can we expect a first-year undergraduate to learn in a single term CS class if they have no previous computing experience? What can we expect to be able to teach any eight-year-old or a 12-year-old about computer science? What are the challenges to teaching computing successfully to students with cognitive disabilities, or who are blind, or who have inadequate mathematics preparation?

We first realized in the early 1980s that we often overestimate what first-time CS students can do.[5] The McCracken Study[2] showed that problems that introductory computer science instructors find reasonable are not solvable by a majority of introductory students.

Part of the trick is simply learning how to measure what students have learned. Briana Morrison and Lauren Margulieux showed[4] that textual programming is cognitively complex. We want students to learn the concepts and skills of programming, but programming itself is a complex activity—asking students to program requires students to be able to do *everything*. Morrison and Margulieux showed they can measure learning toward programming using Parsons Problems.[3] A Parsons Problem asks students to solve a programming problem, then gives them all the lines of code that solve that problem, on tiles or "refrigerator magnets." They showed they could tease out differences in students' understanding of programming, even when the students could not successfully program the solutions yet. Being able to measure the development of these skills, without requiring students to master the skills, is critical to developing learning progressions.

If we want to teach computer science at younger ages, we have to figure out how to reduce that complexity. We

While computer science is now part of STEM in the U.S. by fiat, students cannot access computer science classes as easily as mathematics and science education.

want students to be able to build programs they find motivating and engaging, without mastering all the skills of textual programming first. David Weintrop and Uri Wilensky have shown that students using blocks-based languages (like Scratch, Snap, and Blockly) make far fewer errors than in textual programming languages. Again, part of their achievement is in measurement. They developed a commutative assessment[6] that allowed them to compare the same concepts in both textual and blocks-based programming languages. We will need many kinds of measures to develop learning progressions for computer science.

Computer Science Is Valued but Misunderstood

Students enter undergraduate mathematics and science classes after years of formal education, so they enter with a good idea about what those fields mean. That's not true for computer science. Even undergraduate CS majors do not know what computer science is. Mike Hewner showed that even undergraduate students who declare a major in computer science only have an unclear idea of what the field is about.[1]

Large-scale surveys in collaboration between Google and Gallup have shown that parents and principals think courses in computer science are about how to use personal computers.[c] The surveys show parents and principals highly value computing, and want more computing education for their children. But the parents and principals mostly do not understand what it is.

Students want computer science, whatever they think it is. Many of them want it because of the economic value of knowledge of computer science. They don't know what it is, but they know it can get them a good job and make them more effective at the job they want.

The CS situation is different from science or mathematics. Contrast the number of coding boot camps available in your area to the number of biology boot camps or algebra boot camps. While having a demand for CS is mostly positive, it creates a strange dynamic in the computer science class. Students demand the "real thing" (which we might interpret as "what will help me in a job"), even if they don't know what that is. For example, students might complain about learning a blocks-based language or using a pedagogical IDE because it's not "real"—even if they are not quite clear what "real" is.

Building the Infrastructure for CS Classes

In many countries and U.S. states, you can learn the number of students taking primary or secondary school reading, mathematics, or science classes. In the U.S., hardly any state can tell you the number of students in their computer science classes at any level, or what is being taught in those courses. (In many states, "computer science" and "computing applications" courses are considered the same.) Because computer science has only recently been declared part of STEM, it has not been tracked like other STEM subjects. We don't know with certainty how much computing education is offered in the U.S. today nor where it is offered, which makes it difficult to plan and grow access to computing education.

We believe (from looking at data about Advanced Placement CS and in those states that do track) that far less than 30% of secondary school students even have the opportunity take a computer science course in the U.S. today, and less than 10% of primary school students. To reach the ubiquity of access to mathematics and science education classes, we need to increase the number of computer science classes and teachers by a magnitude. That is an enormous change with dramatic implications. What we have today may not tell us much about tomorrow. The preparation, abilities, and preferences of existing computer science teachers may not be predictive when we have a 10-times-larger population of teachers. We have to invent whole new teacher education programs.

Steps Toward Pervasive Computing Education

While computer science is now part of STEM in the U.S. by fiat, students cannot access computer science classes as easily as mathematics and science education. Many countries are ramping up computing education, so the situation is going to change. As it does, we will have to develop more accurate expectations of how students learn CS, improve our ability to measure learning in computing, develop learning progressions, and create an infrastructure to develop teachers and track progress as we reach the pervasiveness of mathematics and science education. **C**

c https://services.google.com/fh/files/misc/images-of-computer-science-report.pdf

References

1. Hewner, M. Undergraduate conceptions of the field of computer science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (ICER '13). ACM, New York, 2013, 107–114.
2. McCracken, M. et al. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin 33*, 4 (2001), 125–140.
3. Morrison, B.B., Margulieux, L.E., and Guzdial, M. Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Neb., 2015), 21–29.
4. Morrison, B.B., Margulieux, L.E., Ericson, B., and Guzdial, M. Subgoals help students solve Parsons problems. Paper presented at the *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tenn., 2016).
5. Soloway, E. Learning to program = learning to construct mechanisms and explanations. *Commun. ACM* 29, 9 (Sept. 1986), 850–858.
6. Weintrop, D. and Wilensky, U. 2015. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (ICER '15). ACM, New York, NY, 2015, 101–110; DOI: http://dx.doi.org/10.1145/2787622.2787721

Mark Guzdial (guzdial@cc.gatech.edu) is a professor at the Georgia Institute of Technology.

Briana Morrison (bbmorrison@unomaha.edu) is an assistant professor of computer science at the University of Nebraska Omaha.