Ubiquity Symposium

MOOCs and Technology to Advance Learning and Learning Research

Limitations of MOOCs for Computing Education: Addressing Our Needs

by Mark Guzdial

Editor's Introduction

Computing education has some significant education challenges today. We aren't diverse enough, and we need to be able to develop more teachers. Despite popular opinion, the current generations of MOOCs don't meet those needs.

Ubiquity Symposium

MOOCs and Technology to Advance Learning and Learning Research

Limitations of MOOCs for Computing Education: Addressing Our Needs

by Mark Guzdial

Computing education has significant challenges. Not only do we need to diversify computing to meet our growing needs, we need to develop many more high school teachers. Both of the flavors of MOOCs that Candace Thille presents in her opening statement have characteristics that limit the extent to which they can be used to meet these needs. MOOCs are a fascinating and promising new educational technology, and they should be developed and explored. MOOCs allow us to continue current practice in education, just more efficiently. We need to develop computing education in new directions, so that we can improve upon current practice.

Computer science has always had a labor shortage. Historian Nathan Ensmenger suggests high-level programming languages (e.g., Fortran and COBOL) were initially developed to address the programmer labor shortage, by allowing scientists and business professionals to write their own programs. Computing education was originally started in order to develop more programmers. But our current education practice can't meet the need expected from industry. Interest in computing is declining among high-school students. Although the number of students taking Advanced Placement Computer Science (AP CS) is slowly rising, the number of AP CS test takers is still the smallest of all the STEM disciplines. But the problem is even bigger than that because, increasingly, computing educators are asked to teach all those others about computing.

An estimate by researchers at Carnegie Mellon University says for every professional software developer in the United States, there are four more professionals who program as part of their jobs but are not software developers. Additionally, there are nine more people who program, but don't know their work is called programming, e.g., spreadsheet creation or database queries. Few of these non-computing professionals ever take a computing course, which hurts

them when they later try to teach themselves. Brian Dorn found in his studies of end-user programmers that their processes for developing their knowledge are uninformed and erratic. They don't know what information is useful to meeting their needs and which isn't, so they waste a lot of time and make a lot of mistakes. As computing becomes more critical to many disciplines (e.g., for computational scientists, engineers, and journalists), there is an increasing need to provide computing education to people who not aiming for an information technology or computing career.

How are we going to meet all the needs for programmers, and also produce computing-literate professionals who can use computer science to inform what programming they need to do? Current practice won't be sufficient. We have two major challenges:

- *To broaden participation in computing.* Computer science in much of the world is predominantly male, and in the United States, is predominantly white or Asian. There are three problems here. First, women and minorities are not getting access to high-paying technology jobs, which is an equity issue. Second, we will have a less diverse set of ideas around the design table for computing technologies. Third, we will not be able to meet expected demands for computing professionals without diversifying computing. I can provide details from the U.S. context. White and Asian males are a shrinking portion of the U.S. population. Code.org and the ACM Education Policy Committee both cite the U.S. Bureau of Labor Statistics, which suggests there will be approximately 1.4 million computing-intensive jobs in the United States over the next 10 years. Multiply that by four to get the number of end-user programmers, and by nine to get the number of people who will be programming but don't know that they're programming. Right now, estimates suggest that we will only have about 400,000 computing graduates over the next 10 years. Simply getting more women or minorities into computing isn't enough. We can't meet the needs of programming and remain dependent on white and Asian males in the United States. We have to shift the demographic ratios, or we can't possibly meet the demand.

- *To provide access to younger students and to those who are not aiming for IT careers.* How do we provide computing literacy to all those other professionals? Around the world, there are efforts to make computing a subject for primary and secondary school students. Cameron Wilson, the ACM's former Director of Public Policy, calls this an "All hands on deck!" problem. We need all computer scientists to help with scaling computer science into primary and secondary schools. Our critical need is for teachers. There are approximately 2,000 AP CS teachers in the United States, for about 25,000

high schools. We don't even have computer science available in the majority of high schools yet.

MOOCs, as they are currently defined and practiced, do not help with either of these goals. Candace's definition of MOOCs is enough to explain why they are limited in meeting our computing education needs. MOOCs (as defined at Wikipedia) provide lecture-like material (typically through videos). These are broken into small pieces, and are presented with interspersed mini-quizzes. There is additional homework. Feedback is provided, either canned (the system knows what's right and wrong) or through peer-evaluation. There is typically some kind of forum for questions and answers, which is a key part of the connectivist MOOC for "nurturing and maintaining connections."

MOOCs have been used effectively to provide continuing education opportunities to well-educated students who want to refresh or extend their knowledge. But our economy does not depend on refreshing knowledge. We need more people who know computing, and MOOCs don't help there.

Who Currently Takes a MOOC?

Daphne Koller, co-founder of Coursera, has argued (in her TED talk) that the value of MOOCs lies in providing educational opportunities to those who might not get those opportunities otherwise. Sebastian Thrun, founder of Udacity, says his MOOCs offer an opportunity so "everyone can become a proficient computer scientist." But the reality of MOOCs today is that they are only reaching those who already get access to education.

If we look at the data available for those completing computing-related MOOCs, it's mostly for professional development. The students are not really using the online forum for "nurturing and maintaining connections."

- edX released data on students who completed their "Circuits and Electronics" course. These were not new students. Eighty percent had taken a comparable course previously; 37% had a bachelor's degree; 28% had a master's degree; and 6% had doctorates.
- The completers for Georgia Tech's first Coursera MOOC, "Computational Investing," were 88.6 percent white or Asian and 91 percent male. More than 10 percent had their Ph.D.s and more than 40 percent had a master's degree. Ninety-two percent read the

online forum, but only 40 percent ever posted. Of the non-completers, only 21 percent ever posted in the forums.

- In Duke's Bioelectricity course, 37 percent of completers had bachelor's degrees, 27 percent had master's or professional degrees, and 8 percent had doctoral degrees. Only 2 percent had not yet completed high school, and only 9 percent were past high school but not yet in college.

- Of the 100,000 people who signed up for the first Udacity CS101, 10,000 completed. Of those 10,000, 3,000 received perfect scores on the final exam. It's certainly possible that the students learned all the content perfectly in the six-week course. It seems more likely that they knew much of it previously.

- A recent study of University of Pennsylvania's first 32 MOOCs found, "The student population tends to be young, well educated, and employed, with a majority from developed countries."

In all of these reports, the students taking MOOCs are primarily from the United States, Europe, and the rest of the developed world. Empirically, existing MOOCs are not reaching a population that is not already well educated, nor are they reaching a diverse population. That's not the same as saying that they can't. In the particular case of computing education, it's unlikely that they can.

How to Teach Computing to a Diverse Audience

We know part of the problem of getting a more diverse computing workforce is a problem of access. Jane Margolis and her colleagues revealed the few computer science courses available at high schools in the Los Angeles Unified School District were mainly offered to more affluent students, who were mostly white or Asian and male. Underrepresented minority students simply could not get access to those classes.

Access is not the problem in undergraduate education. Computer science is available to all students on campuses in the United States, but these classes are less than 20 percent female on average. The share of undergraduate computing seats to underrepresented minorities is far less. Why is undergraduate computer science so white and male?

Joanne Cohoon has been studying the problem of women in computing for decades. She has found departments can get women to persist in computing, with attention to "the extent to

which faculty members encourage students to persist, mentor for diversity, and expect student success to result from home-work and academic focus" [1]. In Joanne's work, having individual faculty encourage *individual* students to persist is key, as well as mentoring to support diversity.

At Georgia Tech we replicated some of Joanne's work, and found that similar factors influenced underrepresented minorities (our research was presented at ICER 2012, and in more detail in a technical report). We surveyed more than 1,400 students enrolled in introductory computing courses across Georgia, and did an inferential analysis to understand which factors most influenced a set of outcomes related to success in computing: satisfaction in choosing to study computing, likelihood to complete a computing degree, and likelihood to pursue a career in computing. Two factors were particularly influential in all three outcomes: student self-report of his or her own ability, and the student's sense of feeling encouraged to continue, e.g., feeling they belong in their department.

We found the following:

- Ability strongly (statistically significantly) predicted all three outcomes.
- Ability strongly predicted all three outcomes for white and Asian males, even when considering a sense of encouragement.
- The link between ability and these three outcomes was no longer significant when encouragement was added to the regression analysis for women and underrepresented minorities. ***Without encouragement, even high ability women and underrepresented minorities were unlikely to persist in computing.***

How could we provide individual attention, encouragement, and mentoring in online classes of 100,000 students? We might imagine individual attention happen in the online forums. But from what we can see of the available demographics, most students never show up there. Unless a student says something in the online forum, there is no way to provide individual attention in existing MOOCs. How do you provide mentoring, personal encouragement, and a sense of belonging to someone who is invisible?

The research results described here are from empirical studies of what works. Of course, someone might invent something new that will work in MOOCs, too. I can't argue that it's impossible to create more engaging MOOCs. They just don't exist today. It is clear today's MOOCs ignore our existing research and practices for drawing more women and under-represented minorities into computing.

How to Teach Computing to High-School Teachers

There are very few programs in computing education at the undergraduate level. There is too little interest today. Our best chance for getting new computing teachers is to help current teachers learn computing.

High-school teachers are a challenging audience for computing education. Most states in the United States classify computer science as a business, career, or technical education subject (according to the ACM Running on Empty report). meaning most teachers considered qualified to teach CS hold teaching certifications in business. These teachers typically have little science and mathematics education, and few have ever had any programming classes. Thus, they have little background knowledge for learning and teaching programming.

Working teachers (called "in-service" teachers) are over worked. They work a full day, then grade and prepare for class outside of that. This is a perfect mismatch for our current practice in computing education.

Klara Benda, Amy Bruckman, and I studied working professionals taking online computing courses. These courses ran like most CS classes, in an apprenticeship model. The teacher (master) presents some topic, perhaps provides an example, and then expects the student (apprentice) to spend hours working at the computer to figure it all out. An apprenticeship model is great for creating practitioners. It's not great for creating teachers.

The informants we interviewed told us about getting stuck. The content wasn't too hard. Our students had adult lives and families, and they couldn't afford the time of an apprenticeship model. They told us stories about getting one comma out of place and wasting hours. They talked about getting confused by the mathematics and finding no help, even when they asked online. We need new models for teaching computer science, beyond apprenticeships.

What might work for teaching teachers about computer science? That's a big question that people are asking all over the world. In England, they are setting up Computing at Schools Hubs where local teachers can meet up, and a Network of Excellence where master teachers can help new teachers. Here in the United States, my colleagues and I have been setting up a Disciplinary Commons for Computing Educators (DCCE) where teachers gather once a month on a Saturday to share their best practices. We have found that it is critical for teachers to develop a community, which leads to a considerable improvement in teacher retention and improvement in their learning. ***In short, our best practice for growing more computer science teachers involves face-to-face meetings for developing community.***

Current MOOCs Don't Help Solve Our Real Problems

MOOCs are a promising new technology for providing lifelong learning opportunities to existing professionals. They are relatively inexpensive, and as technology improves, MOOCs may become more effective and reach a broader audience. Although they reach students who are already well educated—mostly in the developed world—MOOCs are not yet effectively reaching a diverse audience nor are they reaching students who do not have access to traditional forms of education.

MOOCs are unlikely to meet all the educational needs that face-to-face education can meet. The research presented here shows how some students need encouragement, a sense of belonging, remediation tuned to their needs, and a community. In particular, the students who need this additional support are those most at risk of not succeeding. MOOCs do not meet their needs.

Current MOOCs may simply be an ineffective educational technology. We know MOOCs have a low completion rate. It may also be true that the majority of those who complete *already* knew the content. MOOCs offer a one-size-fits-few model that does not change for individual students. (MOOC developers hope that it will one day, but it doesn't now.) The MOOC model filters for and certifies those who can learn on their own. The role of public education in society is to teach everyone, not just those autodidacts who can learn in a MOOC.

If the only educated people in our society were the ones who wanted to learn the topic (at the start, from the beginning of a class), our society would collapse. We would have too few educated workers to create innovations and maintain the technology we have. Our society depends on teachers who motivate students to persevere and learn.

MOOCs may be bringing the American university to an end—a tsunami wiping out higher education. Given that MOOCs are least effective for our most at-risk students, replacing existing courses and degrees with MOOCs is the wrong direction. We would be tailoring higher education only to those who already succeed well at the current models, where we ought to be broadening our offerings to support more students.

Computer science owns the MOOC movement. MOOC companies were started by faculty from computing, and the first MOOC courses were in computer science. One might expect that our educational advances should address our educational problems. In computing education, our most significant educational challenges are to educate a diverse audience, and to educate non-IT professionals, such as teachers. MOOCs are unlikely to help with either of these right now—and that's surprising.

The allure of MOOCs for computer scientists is obvious. It's a bright, shiny new technology. Computer scientists are expert at exploring the potential of new computing technology. However, we should be careful not to let "the shoemaker's children go barefoot." As we develop MOOC technology, let's aim to address our educational problems. And if we can't address the problems with MOOC technology, let's look for other answers. Computing education is too important for our community and for our society.

References

[1] Cohoon, J. M. "Just Get Over It or Just Get On with It." In Cohoon, J. M. and Aspray W., editors. *Women and Information Technology: Research on Under-Representation*. MIT Press, 2006

About the Author

Mark Guzdial is a professor in the School of Interactive Computing at Georgia Institute of Technology. His research is in computing education, and he maintains the Computing Education blog. He was co-recipient of the 2010 ACM Karl V. Karlstrom Outstanding Educator award, and recipient of the 2012 IEEE Computer Society Undergraduate Teaching Award.

DOI: 10.1145/2591683

