Computing Education as a Foundation for 21st Century Literacy

Mark Guzdial
University of Michigan, Computer Science & Engineering
Ann Arbor, Michigan
mjguz@umich.edu

ABSTRACT

Teaching programming as a way to express ideas, communicate with others, and understand our world is one of the oldest goals for computing education. The inventor of the term "computer science" saw it as the third leg of STEM literacy. In this talk, I lay out the history of the idea of universal computational literacy, some of what it will take to get there, and how our field will be different when we do.

CCS CONCEPTS

• **Social and professional topics → Computing education**; • **Applied computing → Interactive learning environments**.

KEYWORDS

computational literacy, CS for All, computational thinking, history

ACM Reference Format:
Mark Guzdial. 2019. Computing Education as a Foundation for 21st Century Literacy. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.1145/3287324.3290953

1 OUR MISSION: "AT ALL EDUCATION LEVELS"

The mission of the ACM Special Interest Group on Computer Science Education[1] is to serve as a global forum for educators about the learning and teaching of "computing" (not just *computer science*) "at all levels." This is a bold and noble mission. One of the earliest goals of computer science was exactly this – to provide computing to teachers and students at all levels.

Alan Perlis, the first ACM Turing Award laureate, argued in 1961 that all university students take a course in "computers," and that they should all learn programming [9]. He argued that programming gave students a new way to see the world and solve problems – within their own discipline. He said, "Given then the appropriate computer, the capability of developing programming systems, the proper freshman course[2], and possibly a good follow-up program, the computer will achieve its ultimate role as handmaiden to scholarly university activities."

[1]https://sigcse.org/sigcse/about/profile
[2]Even Perlis saw the challenge of computing education as mostly about CS1.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02.
https://doi.org/10.1145/3287324.3290953

Like Perlis, Seymour Papert was interested in the computer as a support for learning activities, especially for children. He made the claim "that children can learn to program and learning to program can affect the way that they learn everything else" [8]. His point is not to teach computer science for its own sake, or as vocational training, but as a support for learning in any discipline.

The earliest reason for teaching students about computers was to support learning in science, engineering, and mathematics (what we now call *STEM*). George Forsythe published the term *computer science* in an article in the *Journal of Engineering Education* [4]. For Forsyth, computers were the third leg of literacy for STEM students. He wrote in 1968 [5], "The most valuable acquisitions in a scientific or technical education are the general-purpose mental tools which remain serviceable for a lifetime. I rate natural language and mathematics as the most important of these tools, and computer science as a third."

Alan Kay [7] and Andrea diSessa [2] used the term "literacy" to describe this use of computation as a medium, like reading and writing. Kay wrote, "Computer literacy is a contact with the activity deep enough to make the computational equivalent of reading and writing fluent and enjoyable." We can study reading and writing for their own sake, but for most of us, reading and writing is what enables us to express ideas, to communicate with others, and to understand our world. Literacy supports and affects how we learn. diSessa and his students developed Boxer as a environment in which students could develop computational literacy for a lifetime.

When I read SIGCSE's mission, I hear the mandate of Forsythe, Perlis, and everyone who has wanted computing to play a role in learning "at all levels." This is our organization's mission, to serve educators who bring computing to all students.

2 WHEN COMPUTING IS LIKE MATHEMATICS AND SCIENCE

Computing education will look entirely different when we teach students at all levels to use computing as a literacy, like reading, writing, and mathematics. We are currently struggling with the overwhelming weight of rapidly rising enrollments [1], but we need to prepare for much greater numbers if we are to meet this mandate. We need to plan for dramatic growth.

Here is one way to think about the scale of that growth. The SIGCSE Technical Symposium draws about 1700 attendees. ACM SIGCSE's International Computing Education Research Conference (ICER) has about 150 attendees. The K-12 CS teachers conference (CSTA) had about 750 attendees last year. Let's consider the difference in scale when we are reaching students "at all education levels." Mathematics and science today is taught to all students, and there are conferences dedicated to supporting those teachers. Here in the US, the National Council of Teachers of Mathematics

had 1,000 attendees at their research conference, and 9,000 at their annual meeting. The National Science Teachers Association has 9–12,000 attendees annually at their national conference. Computing education may not ever look exactly like science or mathematics, but the scale to reach *everyone* is likely comparable.

Computing education will need decades to reach this scale. We are a young discipline, at 50 years old. In contrast, the American Society of Engineering Education (ASEE) recently celebrated their 125th year. The National Council of Teachers of Mathematics was started in 1920, so it soon will be 100 years old. At SIGCSE's 100th anniversary, we may still be talking about achieving Forsythe's vision of computing as a literacy alongside text and mathematics.

3 CHANGING COMPUTING EDUCATION

Meeting the needs of students "at all education levels" is going to require changes in what we teach and how we teach. My thinking about the core principles of computing education has been shaped by an award-winning paper at ICER 2017 [10]. Katie Rich and her co-authors defined learning trajectories for K-12 computer science concepts, but their work is not limited to elementary and secondary school students. Their goal was to identify the most fundamental ideas needed to learn programming. Some of those ideas are:

- Precision and completeness are important when writing instructions in advance.
- Different sets of instructions can produce the same outcome.
- Programs are made by assembling instructions from a limited set.
- Some tasks involve repeating actions.
- Programs use conditions to end loops.

For most of us, these are so automatic that we may not even explicitly teach them. However, students (at all levels) do not know these ideas and need to be taught them. Even subsets of these fundamental ideas are important and powerful.

Computing education means different things to different audiences [6]. Millions of children use the programming language Scratch, but most use only a small part of Scratch's capabilities. For example, a great many Scratch projects use no repetition at all, or never use conditionals to end loops [3]. Computing is so powerful that you can say something useful and interesting without even including everything on Katie Rich's list. Literacy does not require mastery or even use of all the concepts of programming. Fluency is the goal, and scholarly interests can be served with only a small piece of computing.

This last summer, I visited the London Science Museum and saw the device in Figure 1, an printing telegraph machine from 1860. It is an anachronism today, not just because it's a telegraph machine. The developers of this device recognized the powerful idea of generating the Morse code for a letter with a keystroke. But the only keyboard they had at hand was a piano keyboard. Printing telegraph machines date from 1840, but the QWERTY keyboard was only patented in 1868. For almost 30 years, if you wanted a keyboard, a piano keyboard was the only option.

Computing education is only 50 years old. We may be in the period before our QWERTY keyboards are invented. Some of what we are inventing may not last long. It may become an anachronism.

Figure 1: An 1860 Hughes' Printing Telegraph Machine.

We have to consider that what we see as computing education may be a great idea with a piano keyboard attached. Things will change.

4 A CALL TO ACTION

We need to make more computing educators, of all kinds, at all educational levels. Some of these teachers will be in science, mathematics, or other non-CS disciplines. Some of them will not know enough to pass our current CS1's. That's okay. Expressive power does not require everything we currently teach in our CS1.

We have to find our allies. I draw on research and practice from physics education research, educational psychology, and learning sciences. Creating a literacy for the 21st century is a large multidisciplinary endeavor.

We need to invent, to mutate, to evolve. We should not just replicate our existing computing education tools, curricula, and classes. We have to make new kinds of computing education to meet the goals of computing literacy at all educational levels. Be on the lookout for the QWERTY keyboards, which may seem awkward at first (and may even be sub-optimal), but will allow us to expand our reach and to grow our field to meet the needs of all students and teachers.

REFERENCES

[1] Tracy Camp, W Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: the growth of computer science. *ACM Inroads* 8, 2 (2017), 44–50.

[2] Andrea diSessa. 2001. *Changing Minds.* MIT Press.

[3] Deborah A. Fields, Yasmin B. Kafai, and Michael T. Giang. 2017. Youth Computational Participation in the Wild: Understanding Experience and Equity in Participating and Programming in the Online Scratch Community. *ACM Trans. Comput. Educ.* 17, 3, Article 15 (Aug. 2017), 22 pages. https://doi.org/10.1145/3123815

[4] George Forsythe. 1961. Engineering students must learn both computing and mathematics. *Journal of Engineering Education* 52 (1961), 177–188.

[5] George Forsythe. 1968. What to do till the computer scientist comes. *Amer. Math. Monthly* 75 (1968), 454–462.

[6] Mark Guzdial. 2018. What We Care About Now, What We'll Care About in the Future. *ACM Inroads* 9, 4 (Nov. 2018), 63–64. https://doi.org/10.1145/3276304

[7] Alan Kay. 1984. Computer Software. *Scientific American* 251, 3 (1984), 52–59.

[8] Seymour Papert. 2000. What's the big idea? Toward a pedagogy of idea power. *IBM systems journal* 39, 3.4 (2000), 720–729.

[9] Alan J. Perlis. 1962. The Computer in the University. In *Computers and the World of the Future*, Martin Greenberger (Ed.). MIT Press.

[10] Kathryn M. Rich, Carla Strickland, T. Andrew Binkowski, Cheryl Moran, and Diana Franklin. 2017. K-8 Learning Trajectories Derived from Research Literature:

Sequence, Repetition, Conditionals. In *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM, New York, NY, USA, 182–190. https://doi.org/10.1145/3105726.3106166