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.

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

DOI:10.1145/3365581

<http://cacm.acm.org/blogs/blog-cacm>

Getting High School, College Students Interested in CS

Mark Guzdial considers how few U.S. high school students take computer science, while Robin K. Hill shares what she's learned in teaching first-year college computing students.

Mark Guzdial
A Minuscule Percentage of Students Take High School Computer Science in the United States:

Access Isn't Enough

September 2, 2019

<http://bit.ly/2oupbk6>

If I told you only 4% of all high school students in the U.S. were taking science or math classes, you'd be aghast. That's such a tiny percentage! If 96% of students were *not* getting science or math classes, you could reasonably argue it does not exist in any practical sense.

Over the last few months, several reports provided new insights about U.S. high school computer science (CS):

- In California, only 3% of 1.9 million high school students enrolled in CS in 2017 (<http://bit.ly/2pq5kTH>).

- In Texas, 3.76% of its students completed a CS course in 2017–2018 (<http://bit.ly/2ox7cK6>).

California and Texas are two largest states based on U.S. population, but we can't generalize to *everyone* based on

those states. We don't have data on who is taking CS across the U.S., due to our state-centric, decentralized model of primary and secondary school education.

California and Texas are among the *leaders* in implementing CS education. Both are members of the Expanding Computing Education Pathways (ECEP) Alliance (<https://ecepalliance.org/>). Reports from ECEP states tell a similar story. For example, in Indiana, the most popular high school CS course enrolled only 2% of students in the state (<http://bit.ly/2oAq6Q6>). Miranda Parker is finishing her dissertation on factors influencing CS adoption at the high school level, and she's found only 1% of high school students in Georgia take a CS course. I suspect states not involved in ECEP have *lower* participation in CS, not higher. Given these numbers, it would be hard to believe more than 4% of U.S. high school students take a CS course.

One surprise for me in these reports: many U.S. high schools offer CS. I wrote a **BLOG@CACM** post in 2012 (<http://bit.ly/2ox5NDI>) estimating about 10% of high schools in the U.S. offered comput-

er science. Today, that number is much larger. In Texas, it is 43%. In California, it is 39%. In Georgia, it is around 40%, and about 1/3 of high schools in Indiana offer CS classes. The high schools that do offer CS are most often in larger (wealthier) high schools. Many students *could* take those classes. But they do not.

A decade ago, I thought the reason so few students were taking CS was access. Today's data paints a different story.

Even if CS is in the high school, few students sign up for it. Those who do are mostly male (71% in California, 74% in Texas). Some states are better at getting underrepresented minorities into CS classes. It's 48% in Texas. In California, only 16% of AP CS A exam takers are members of underrepresented groups.

Some states might decide this is not a problem; 3% may be enough. U.S. universities still reeling from overwhelming numbers of students in CS classes (<http://bit.ly/2oCXoOs>) might not *want* more students to become interested in CS at the high school level. I hear from schools that *most* introductory CS students they get have already had high school CS. These are compatible numbers. There are 15.3 million high school students in the U.S.; 3% of that is around 450,000 students—which I expect is more than the number of students in university intro CS in the U.S. If CS participation numbers increased at the high school level, probably more students would want to take CS at the university level (even if not as a CS major). Universities would have to restructure to manage such an enormous load.

Both the U.K. Computing at School

movement (<http://bit.ly/2oGcQJF>) and the U.S. CS for All movement (<http://bit.ly/2n4WCJS>) argue everyone deserves to learn about CS. If you buy their argument, these are disturbing numbers. We need a different strategy than simply getting CS into high schools. Now, we have to get students into the classes.

Comments:

South Carolina recently revised its high school computer science standards to require computer science beyond just keyboarding.

<http://bit.ly/2oDOMZG>

James Burton

Robin K. Hill Lessons from a First-Year Seminar

July 27, 2019

<http://bit.ly/2mZZuYm>

I teach a first-year seminar called “The Beauty and Joy of Computing” (after a course at Berkeley).³ All freshmen are required to take one of these seminars, offered in different subjects; the general pedagogical goal is teaching them to be college students. I taught it a year ago, and have a few reflections in preparation for a new round.

I am proud to work at a land-grant university with almost open enrollment, which requires careful treatment of disparate student backgrounds. The challenge presented by the computer science (CS) version of this course is that student interest in computing ranges from nil to avid, and student experience with computing ranges from none to expertise (the expertise ranges from narrow to haphazard). I do not pretend to have resolved these into a harmonious program of study, but I can sketch some of my attempts.

To introduce first-year students to the college environment, the course includes a research paper, group projects, and oral presentations. They need to learn to distinguish peer-reviewed results from public relations, and scholarship from argumentation and opinion. They need to be able to explain AI, Internet, and general hardware and software concerns to their peers, their families, and the public.

Major Components

Coding

I hold the pedagogical principle that in every class, students should come away

with a body of knowledge. In this case, it's a few basics of CS. We code or complete rudimentary Java programs with the BlueJ development environment. We look at number systems, character codes, encryption, compression, and similar topics up to the Fundamental Theorem of Arithmetic. Components that did not work for us include the block-style programming platform Snap!. I prefer beginners see plain source code in a text file, for the standard view of how computers are programmed.

Internet Issues

To engage students whose strengths lie in areas other than programming, we review contemporary Internet issues via the 2008 textbook by Abelson, Ledeen, and Lewis,¹ supplemented by journalism from *The New York Times* and other sources. A list of issues serves as the sign-up choices for the group project.

Literature

I believe young people interested in computers should also learn about the humanities (and so should other young people). I assign the reading of a classic novel that shares one of the themes we study. Last year, the country read *Frankenstein* in honor of its 200th anniversary. We joined in, discussing it as an account of the unintended consequences of technology,² notwithstanding that a 19th-century novel consisting mostly of lengthy soliloquies on guilt and misery is heavy going, with no actual technology.

Other classics that parallel issues of modern technology (besides 1984 and surveillance) include *The Scarlet Letter* and public shaming. Many great works of science fiction reflect tech issues, but I want a distant rather than close parallel, for purposes of abstracting the common theme. Suggestions are welcome!

General Pedagogy for Freshmen

Here are mixed tips for teachers, organized only by idiosyncrasy:

1. Don't cover up the difficult, tedious, fussy bits. Discuss input and output; variable declarations; defaults and parameters on the installation of software. These are counterintuitive protocols people must face to master computers.
2. Expose critical points of misunderstanding. Give exercises that distinguish

between $A[i]+1$ and $A[i+1]$; that interpret 11010011 as a char, number, and instruction; that contrast variable names such as *Temperature* and *temperature*.

3. If you mention UTF-8, or the Fundamental Theorem of Arithmetic, or a substitution cipher, tell them what it is right then and there. You can return to cover it comprehensively later.

Group project: To preclude students working with friends, have them form teams with classmates not already in their phones. Require meeting schedules and regular updates.

Research paper: Give incremental assignments, with suggestions incorporated into the next draft. Distinguish between objective journalism and research papers as acceptable sources, marketing and white papers as unacceptable.

I still seek suitable research publications. What journals or magazines provide peer-reviewed articles on CS for the neophyte? Our library subscribes to ACM and IEEE publications, but online. For browsing and discovery, I direct students to special technology sections or issues of major newspapers and magazines. Again, suggestions are welcome!

Teaching at the novice level invokes standard areas of philosophy:

- Metaphysics, in extracting the essence of the subject.
- Epistemology, in considerations of how to convey the essence of the subject.
- Ethics, insofar as the teacher answers the question of the subject, “What good is it?”
- Aesthetics, in how to make it attractive to learners. This may be a sly application rather than a scholarly endeavor.

Let's do our best for our students, whoever they are.

References

1. Abelson, H., Ledeen, K., and Lewis, H. *Blown to Bits: Your Life, Liberty, and Happiness after the Digital Explosion*. 2008.
2. Hill, R.K., “FictionStein,” BLOG@CACM, November 21, 2018, <http://bit.ly/2oEKyza>
3. The Beauty and Joy of Computing. University of California, Berkeley, Advanced Placement course for late high school and early college students. Accessed 25 July 2019.

Mark Guzdial is a professor in the Computer Science & Engineering Division of the University of Michigan, USA. Robin K. Hill is a lecturer in the Department of Computer Science and an affiliate of both the Department of Philosophy and Religious Studies and the Wyoming Institute for Humanities Research at the University of Wyoming, USA.

© 2019 ACM 0001-0782/19/12 \$15.00