2008-08

The Profession of IT, Voices of Computing

Denning, Peter J.

Voices of Computing (August 2008) The choir of engineers, mathematicians, and scientists who make up the bulk of our field better represents computing than the solo voice of the programmer.
http://hdl.handle.net/10945/35481

DOI:10.1145/1378704.1378711

Peter J. Denning

The Profession of IT
Voices of Computing

The choir of engineers, mathematicians, and scientists who make up the bulk of our field better represents computing than the solo voice of the programmer.

Although enrollments in computing degree programs appear to have bottomed out at approximately half of their 2001 level, there is no reassuring upward trend. The industry need for computing professionals will continue to exceed the pipeline by at least one-third for some time to come. Why do low enrollment rates persist in such a good market?

Several key factors influencing low enrollment rates are connected to the myth "CS=programming"—tales of dwindling employment opportunities, negative images of computing work, and inflexible curricula.[2,3] Reversing this myth can result in considerable progress.

Thirty-five years ago, Edsger Dijkstra reacted to his generation's version of this myth by declaring himself proud to be a programmer.[5] Many followed his lead. ACM has been proud: half the A.M. Turing Award winners are in programming, algorithms, and complexity.[3] But our internal self-confidence did not dispel the external myth.

Twenty years ago, the ACM and IEEE warned that the myth could become damaging.[4] Today, the word "programming" itself generates misunderstandings. Internally, it is broad: design, development, testing, debugging, documentation, maintenance of software, analysis, and complexity of algorithms. Externally, it is narrow: the U.S. Bureau of Labor Statistics defines "programmer" to mean "coder." Often without realizing it, insiders and outsiders interpret the same words with entirely different meanings. When insiders broadened to object-oriented programming, outsiders thought we narrowed to the Java language.

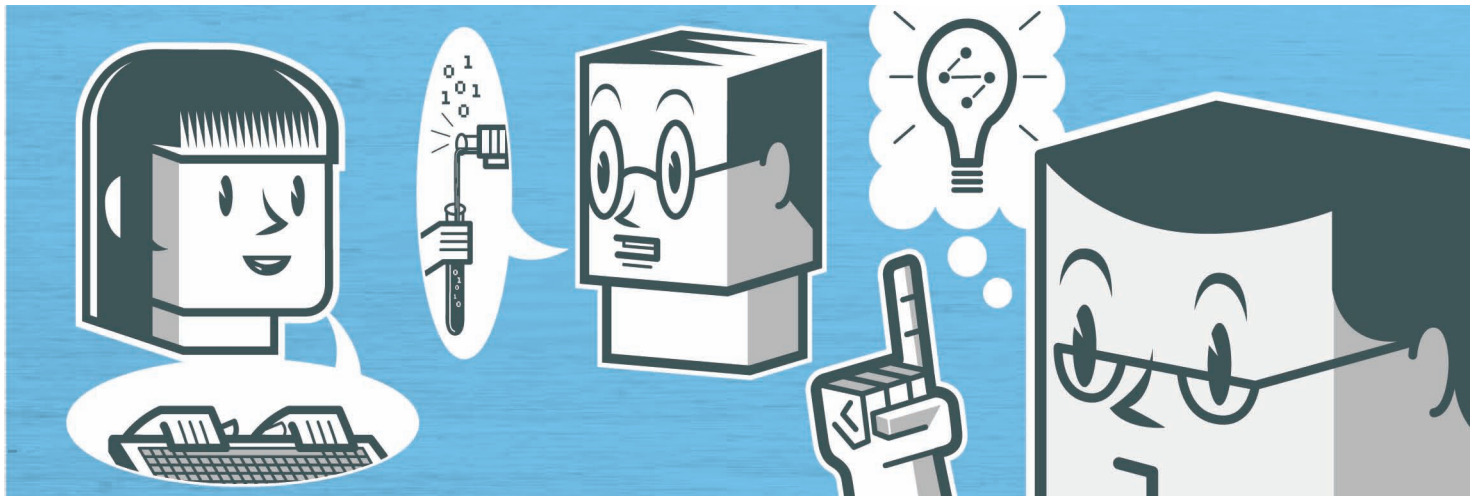
Ten years ago, we tried another tack. We broadened our view of computing to include information technology (IT),[1] and we defined what it means to be fluent in IT.[7,8] These works were embraced in high schools and helped generate enrollments in IT but not CS. They have not dispelled the myth.

Today, Clay Shirky notes a trend that may help explain the durability of the myth.[6] The general public is now confronted with an amazing array of powerful tools for the common computational tasks. Many believe they can accomplish what they need as amateurs. Only a few professionals are needed to program all these tools for the many. There is no special attraction to being a professional.

How can we communicate the richness of our field and dispel the myth? What if we learn to speak in the voices of the many kinds of computing professionals? The programmer is a solo voice. The whole, loud choir could dispel the perception that the bulk of computing is about programming. The choir might also help make professionalism more attractive by showing our many critical specialties that cannot be done by amateurs.

To speak in a professional's voice, I immersed myself in the professional's practice. I spoke of war stories, experiences, ambitions, fears, and everyday things. I sang the joys (and sorrows) of being a professional.

Education philosophers such as John Dewey maintained there are two ways of learning, which can be called "learning-about" and "learning-to-be." Learning-about means to acquire a description; learning-to-be means to acquire the practice. Learning about carpentry, music, or programming is not the same as being a carpenter, a musician, or a programmer. Programming seems to blur this distinction because programmers build descriptions of al-



gorithms. Programming predisposes us to the "about" side of computing. We are more used to speaking about the principles and ideas of our field than about how individuals experience them. Descriptions of computational methods can be dull and lifeless compared to war stories from professionals who design and use them.

What follows are six voices of computing professionals. I added a seventh, non-professional voice, which I call the Last Voice. It is last not only because it appears at the end of this column, but because it may be the last voice consulted by young people before deciding against computing as a major.

All these voices are already within you. Except the last, just let them speak.

The Programmer

I love programming. I know a lot of languages and can make computers really hum. I do my best work when no one bosses me around—that's when I am at my most creative. You know, programming is the most fundamental part of computer science. No computer can run without a program. I enable everything else in computing. I have written some history-changing programs. Just think about the software in the Apollo missions—I helped get us to the moon. Think of all those multiplayer virtual reality games—I give a lot of people immense pleasure learning important skills and shooting each other up. I get you safely across the country by helping the air traffic controllers. I get you your food by helping to route the trains and trucks. I gave you your word processor, spreadsheet, PowerPoint applications, and even a few friendly hearted Easter Eggs. I attacked the Internet with a worm in 1988 and then helped stop the worm and catch the perp. I do a lot of things for you. I know that sometimes you look down at programmers and sometimes you think of us as the computer science equivalent of hamburger-flippers. But we deserve your respect and admiration.

The User

I love using computers. I'm not a computer scientist, and I don't want to be. I just love using the stuff computer scientists make. Awesome! I get some really spiffy things done with your tools even though I am an amateur. Most of the time, your stuff does not bankrupt me, waste my time, or kill me. My cell phone, instant messages, Web, Internet, Google Earth, Microsoft Office, iTunes, iPod, and the ACM Digital Library. It just goes on and on. I am so grateful to have all this computer stuff. My wants and needs determine what computer scientists can sell, so they often listen to me very carefully. Without those wants and needs, in fact, I'd be a nobody.

We are more used to speaking about the principles and ideas of our field than about how individuals experience them.

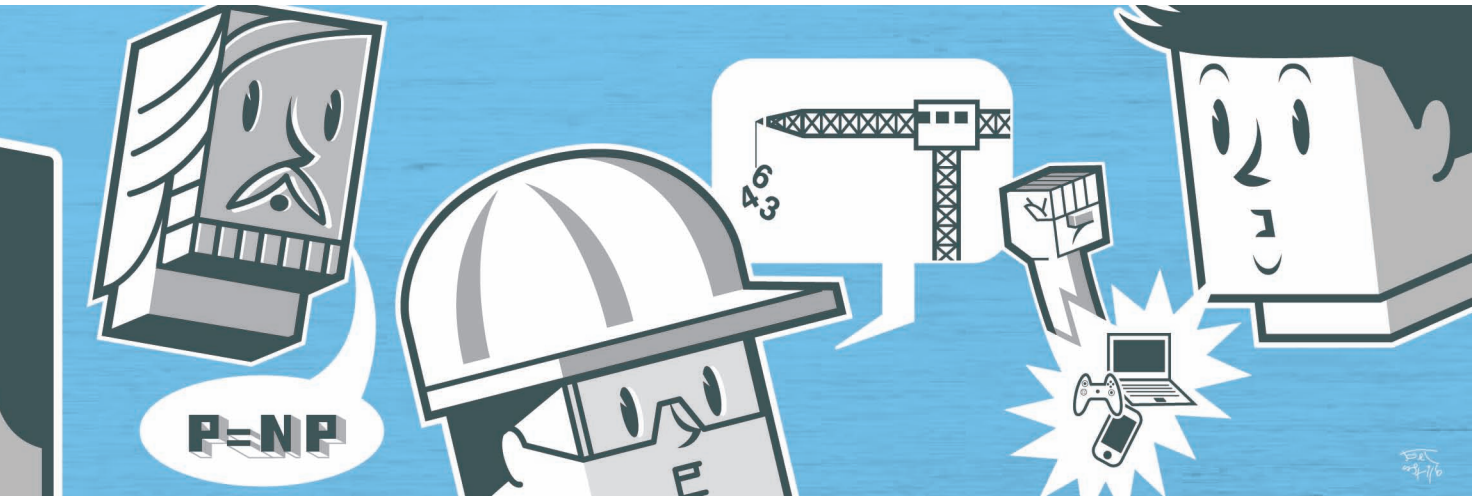
The Computational Thinker

I love problem solving. Not just any old problem solving, but problem solving using algorithms. I love finding ways to apply algorithms I know to solve problems that folks didn't realize could be solved. It's such a powerful way to solve problems. All you have to do is think algorithms and—poof!—solutions appear. Sometimes I implement those solutions myself, and sometimes I let my friends the programmers do that. I've helped biologists search DNA databases, meteorologists forecast weather, petrologists find oil, oceanographers track ocean currents, linguists teach languages, and tax collectors insert spreadsheet algorithms into the law. Every so often somebody asks if I am a computational scientist. I answer no—while I think about how algorithms can help scientists, I don't do their science for them. I'm all about thought. One of my greatest successes is to get politicians to think that through their laws they are programmers of national social systems. I've got economists thinking they can program the economy with the right policies. Perhaps my greatest triumph is to get people everywhere to think their brains are computers and that everything they do and say is simply an output.

The Mathematician

I love mathematics. I know mathematics sounds pretty abstract to a lot of people. It's not for everyone. We've long been recognized as the language of physics. Now we've got the addition-

ILLUSTRATION BY JOEL CASTILLO



al recognition of being the language for computation. We help people find representations for real-world things and then prove things about their representations. In computer science we have invented new mathematics for analyzing algorithms. We explained why a binary search is so much faster than a linear search, and why a bubble sort is so much slower than a merge sort. We figured out how to parse programming languages efficiently, making the jobs of programmers so much easier. Our tools help programmers prove that their complicated programs actually work, meaning that users can sleep at night knowing that their airplanes won't crash and their business software won't ruin them. Our biggest triumph has been to show that over 3,000 common problems in science, engineering, and business are so difficult to solve that even the fastest supercomputers would take centuries on simple versions. We call this the P=NP issue. Whoever proves that P=NP would win all the math prizes and the ACM Turing Award. And no, proving P=NP does not boil down to proving N=1.

The Engineer

I love building things. My math friends like picturing things in their minds; I like holding things in my hands and putting them through their paces. You tell me what you want, what budget I have, and how much time I have, and I'll find a way to build a computing system that does it. I don't need everything to be figured out mathematically before I can start. I built your operating systems, your networks, your TCP and IP, your air traffic control system, your banking systems, your game engines, and your search engines. I built your memory chips, your CPUs, your stacks, your graphics displays, your warehouse computers, your BlackBerries, and your iPods. I know how to make software and hardware artifacts reliable, dependable, usable, safe, and secure. I love the smells of solder, motherboards, routers, power supplies, and musty cable racks. Sometimes I even think I can smell rotting bugs in software. I'm so good at doing things faster, cheaper, and better that I keep on giving you Moore's Law year after year.

The Scientist

I love discovering new things about nature. Recently my friends in biology have discovered that DNA transcription is a natural information process. What an amazing discovery. Computation is not an artifact of a computer, it's part of life! My friends in physics, economics, materials, chemistry, meteorology, oceanography, and cosmology are all making similar discoveries. What a great time for collaborations on new discoveries about those natural processes. But that's not all I do. I discovered scientific principles for computing. My scientific analysis guided the design of the first electronic computers. My principle of locality helped us achieve high performance through caching in everything from chips to the Internet. I discovered fast algorithms for throughput and response time of large systems and networks, launching the performance evaluation industry. I brought the experimental method to architecture, program performance improvement, large system design, mathematical software, modeling, and simulation. My greatest triumph in the CS realm has been with artificial intelligence. Now that they have accepted my methods, they are making remarkable advances with machines that mimic human intelligent behavior.

The Last Voice: The Catalog

Students begin by learning the use of computer programming as a problem-solving tool. Topics in procedural programming include expressions, control structures, simple data types, input-output, graphical interfaces, testing, debugging, and programming environments. The student then advances to problem solving with object-oriented programming. Topics include classes, inheritance, packages, collections, exceptions, polymorphism, and recursive thinking. A good deal of time will be spent on programming projects. C

References

1. Denning, P. Who are we? *Commun. ACM 44*, 2 (Feb. 2001), 15–19.
2. Denning, P. The field of programmers myth. *Commun. ACM 47*, 7 (July 2004), 15–20.
3. Denning, P. Recentering computer science. *Commun. ACM 48*, 11 (Nov. 2005), 15–19.
4. Denning, P. et al. Computing as a discipline. *Commun. ACM 32*, 1 (Jan. 1989), 9–23.
5. Dijkstra, E. The humble programmer. *Commun. ACM 15*, 10 (Oct. 1972), 859–866.
6. Shirky, C. *Here Comes Everybody*. Penguin, 2008.
7. Snyder, L. *Fluency with Information Technology*. Addison-Wesley, 2002.
8. Wing, J. Computational thinking. *Commun. ACM 49*, 3 (Mar. 2006), 33–35.

Peter J. Denning (pjd@nps.edu) is the director of the Cebrowski Institute for Information Innovation and Superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.