Peter J. Denning and Andrew McGettrick

Recentering Computer Science

The recent decreases of enrollment in computer science programs signal a chasm between our historical emphasis on programming and the contemporary concerns of those choosing careers.

Rising crescendos of voices sound alarm about the sharp downturn in young people selecting a computing field for their careers [10]. In the U.S. this is true for computer science, computer engineering, information technology, information systems, and software engineering. In computer science, the numbers of incoming freshmen fell by 60% between 2000 and 2004. The percentage of all college freshmen planning a major in computer science dropped to 1.4% in 2004, down from its peak of 3.4% in 1998 and below a trough of 1.6% in 1992–1993 [7]. To compound the problem, internal drop rates of 35% to 50% are common.

Meanwhile, IT job projections are positive. The U.S. Bureau of Labor Statistics forecasts job growth in all computing specialties of 20%–50% by 2012, except for computer operators (decline) and programmers (flat). Finding people to fill the growing number of IT jobs will soon be more difficult than ever.

In the U.K. a similar trend is evident. Between 2000 and 2004 the numbers of applicants for computer science programs went from 24,151 to 13,715 and for software engineering programs from 1,892 to 917. They also have the problem of high numbers of dropouts. Over the same period the numbers applying for mathematics programs went from 3,925 to 4,533 and for electronic and electrical engineering programs from 3,061 to 5,852. While these related fields are smaller, their trends are in the opposite direction.

In other European countries also, CS enrollments are dropping while jobs are rising. This is happening even in countries where job outsourcing is not an issue.

Are these numbers part of the normal ebb and flow of labor markets? Will the growing number of IT jobs eventually draw students back, as in the past? It does not appear this way. Surveys of high school students reveal that many of them find fields such as bioinformatics and molecular biology more glamorous; many believe that opportunities for good careers in computing were lost in the dot-com bust and will not return as jobs migrate to cheaper-labor countries [7].

Diagnosis

The plummet has been blamed on various factors: belief in job loss; media portrayals of computing as stodgy and nerdy compared to other fields; an impression that computing requires extraordinary proficiency at math; uninformed high school

PETER HOEY

counselors; and a 2001 NCAA directive that high school students cannot use computing courses to satisfy initial eligibility for college athletics. However, except for the first, these factors also existed during CS boom-years—they are not convincing.

We believe that a deeper issue lurks behind these factors. It is the persistence of the image that computer science is a field of programmers [4]. Most computing people understand "programming" to mean the discipline concerned with design, development, testing, debugging, documentation, and maintenance of software. Along with analysis of algorithms and complexity theory, many of us take programming as the heart and soul of computing. Half the ACM A.M. Turing Award winners are in these categories (see Table 1).

Starting around 1985, some of us began to warn that our external image, "CS=programming," conveyed too narrow a view of the field [4]. Since that time, the Bureau of Labor Statistics has co-opted the title "programmer" to mean something like "coder" and not the full range described in the preceding paragraph. The result has been a narrowing of the external understanding of the field. In this light, what we internally thought of as *broadening* to object-oriented programming was perceived externally as a *narrowing* to the Java language.

Given the narrowing of external perception, it is easy to understand a prospective student's lament: "If programming is the heart and soul of computing, and is being auctioned off to the lowest offshore bidder, what is the future for me?" Microsoft's Bill Gates made an extraordinary tour of several U.S. universities in February 2004 to tell students that there are numerous computing jobs beyond programming that will never migrate offshore. The publicity surrounding his trip did little to reverse the decline.

theory, complexity	10
programming, compiling	9
analysis of algorithms	4
numerical analysis	2
networked systems	5
artificial intelligence	6
database	3
operating systems	3
cryptography	3
architecture	1
communications	1
graphics	1
software engineering	1
TOTAL	49

Table 1. ACM A.M. Turing Award winners 1966–2004.

Our problem is strikingly similar to the pattern described by Geoffrey Moore in *Crossing the Chasm* [1, 9]. A large communication gap separates us from the masses who want to use computing technology. To cross it we need to learn to speak to them about things they care about.

ACM's Approach

ACM is mounting a three-pronged approach to reverse this amazingly narrow impression of computing. (1) The new Computer Science Teachers Association will be taking the message directly into high schools and working to make high school computing courses much more interesting than Java programming. (2) ACM will seek greater visibility for ACM events, speak out on contemporary issues, raise awareness of the health of computing, and call attention to the optimistic projections for computing careers. (3) The Education Board will endorse experiments to find curriculum formulations that express better what computing is about and are more appealing to prospective students.

In what follows we will speculate about such a formulation and encourage experimentation with it. Our idea is influenced by the ACM Education Board's Great Principles of Computing project, which is seeking a portrayal of our field in terms of our fundamental principles and core practices [2]. Programming is one of four core practices; the other three are systems thinking, modeling, and innovating. We believe that giving innovation a more prominent role in our curricula would go a long way toward improving our image and our appeal.

Even a small change in this direction can pay off. At Georgia Tech an introductory course on multimedia computation attracted large numbers of students, over half of them women, and had a retention rate of over 97% [11]. The course allowed students to express deep creative urges while learning solid science.

A Theme of Innovation

The notion that *innovation* is central to a study of computing seems extremely important. Innovation is the creation and adoption of new practices supported by information technology [3]. Innovation is not simply the invention of novel technologies; it must include changing practice within organizations and communities. It can take various forms such as new products, new processes, new functionality, new research insights, or new business models. Innovation is a clear theme in government and industry, where it is seen as a source of wealth, competitiveness, and professional success:

Harnessing innovation in Britain is key to improving the country's future wealth creation prospects. [8]

… there is a clear need for the business managers and leaders of the future … to grasp the strategic implications of IT and be able to innovate through effective IT exploitation. [6, p. 49]

If you're not bringing new ideas to the table, you're signing your own pink slip. [12]

Innovation is neither a science nor an art. It is a practice. [5, p. viii]

It thus appears that an innovation theme would resonate with industry needs.

Although there are a few gifted people who seem to have a talent for repeated innovation, innovation is seen mostly as unpredictable, a matter of luck which ideas will make it. In our previous work, we have proposed a more imaginative view of innovation as the result of certain foundational practices [3]. Innovation is not the province of the few; it is key to the role of professionals in IT. It is not a matter of luck or special talent; it is a skill and a practice.

The important point for the current discussion is that innovation can be learned and a spirit of innovation can permeate throughout our courses, starting from the first year and building up expertise in innovation through to the final year. Starting early would help win new hearts and minds to computing. We believe the prospect of participating in and causing innovations is highly appealing to students who want to make a difference in the world. How can this be done?

The first challenge is to embed the foundational practices of innovation into the curriculum, so that students learn innovation by doing, without necessarily being aware they are engaged with systematic processes. The intention is that innovation should become an essential aspect of their attitude of mind. Seeking opportunities for innovation can become a way of life for students, ingrained from the very start. We would aim to instill a habit of innovation.

As a start, we would add the study of great innovations. The stories of innovators, and how and why they did it, can be educational and inspiring if done well. We would organize the study of an innovation into three parts: (1) the situation in the period preceding the invention, characterized by ad hoc solutions to pressing problems, then (2) the innovator's discovery of a principle and how it worked, and then (3) widespread integration of the principle into well-behaved systems down to the present day. This places great principles in context: we would immerse students in the problem and help them invent the principles for themselves. We would help students follow in the footsteps of great innovators by literally retracing their paths. After enough imitation steps they will start innovating on their own.

The innovation themes that might be learned during a student's four-year undergraduate education could be divided in two parts: *creation* of ideas (Years 1 and 2) and *adoption* of ideas (Years 3 and 4). In the beginning innovation appears technical, but later it is about changing organizations and changing the perspectives of individuals.

Year 1: Start learning great innovations of computing; the great principles narrative approach can provide a framework [2]. Learn how to recognize and cherish an opportunity. Learn to recognize the various forms of innovation.

Year 2: Continue with great innovations. Learn how innova-

We view this as an opportunity for professional development and subsequent greater innovation in research on a considerable scale.

tors envisioned and declared new possibilities, committed themselves to one, and then adjusted to people's reactions.

Year 3: Learn to identify settings in which innovation is possible; evaluate value and worth; show how to get to the result; evaluate risk. Plan a project for an external community, find a sponsor, listen to concerns, modify the plan, and develop a prototype.

Year 4: Work with the external community to adopt the new system and sustain it in their environment.

Exercises and simulation games would be included in some courses to help students learn to do the basic moves for themselves. We would choose exercises and course work carefully to build up student confidence in their own ability to innovate. (It is perhaps worth remarking that such activity is less prone to plagiarism.) We would place a premium on innovation when we assess student work, and devise suitable rewards.

Most academic curricula are arranged around a set of three-hour courses. We suggest that, by creating a category of half-course modules, there would be more flexibility to provide innovative topics and course titles. Table 2 shows some possible course and module titles for the first two years. A lot of current material could be repackaged and included in such modules.

First year courses and modules	
Programming and multimedia	course
Great innovators in computing	course
Computers in support of space travel	module
Building your own computer	module
Securing your computer	module
Robots	module
Second year courses and modules	
Building search engines and other software tools	course
Great innovators in computing	course
Forensics	module
Puzzles and logic	module
The Web and digital libraries	module
Computer graphics and animation	module
Other possible modules	
Benefiting from e-learning	
Biological computing	
Choosing a computer	
Environmental computing	
Health computing	
E-government	
E-commerce	
Mobile computing	

Table 2. Innovation themes in freshman–sophomore courses.

The idea of half-course modules may appear as a stumbling block to some readers. After all, schools have invested millions in their current systems. We acknowledge the challenge. Each institution would have to find its own approach to solving this one. But they should solve it.

The critical point is to set the material in an appealing context, suggested in the titles of the courses. This idea applies also to aspects of mathematics and algorithms; these need to be set in a context that appeals and motivates. We have no doubt there would be challenging work in developing such courses. We do need to change things.

Compare the courses and modules shown in Table 2 with the traditional freshman-sophomore curriculum:

- Computer science I (introduction to programming)
- Computer science II (data structures)
- Computer science III (fundamental algorithms)
- Computer architecture
- Discrete math
- Computing ethics

Conclusion

We have speculated about adding an innovation theme to curriculum that would be very attractive to prospective students.

There are many challenges in such an approach. Core content would need to repackaged and placed alongside innovation material and practice in the new courses. Areas such as computer graphics, concepts from broadcasting, safety-critical or high-integrity systems, multimedia, genomics, nanomaterials, quantum computing, and tools such as Apple's iMovies, provide levers to make interesting choices.

Many faculty and staff would need to learn the practices of innovation themselves before teaching them. We view this as an opportunity for professional development and subsequent greater innovation in research on a considerable scale.

In the 1960s the mathematics community, seeking to make mathematics more attractive as a major, invented the "new math." The initiative failed. Historians cite two prime reasons: the new math was too abstract for students to see connections with their lives, and math teachers were not adequately prepared and thus could not awaken a spirit of joy and adventure in students.

We must be careful to avoid this mistake in responding to the enrollment crisis. The Education Board can encourage interested schools to experiment with new approaches. Those schools can design innovation-themed curricula and everyone else can learn from the experience.

Part of the solution to the current crisis will involve changing the narrow external image that we are a field of programmers. We hope the innovation approach described here will provoke discussion about a new way. C

References

1. Denning, P.J. Crossing the chasm. *Commun. ACM 44*, 4 (Apr. 2001), 21–25.
2. Denning, P.J. Great principles of computing. *Commun. ACM 46*, 11 (Nov. 2003), 15–20.
3. Denning, P.J. The social life of innovation. *Commun. ACM 47*, 4 (Apr. 2004), 15–19.
4. Denning, P.J. The field of programmers myth. *Commun. ACM 47*, 7 (July 2004), 15–20.
5. Drucker, P. *Innovation and Entrepreneurship*. Harper Business (1985).
6. E-skills U.K. and Gartner Consulting. *IT Insights: Trends and U.K. Skills Implications*. November 2004.
7. Foster, A. Student interest in computer science plummets. *Chronicle of Higher Education 51*, 38 (May 27, 2005), A31.
8. HM Treasury, Department of Trade and Industry, and Department for Education and Skills. *Science and Innovation Investment Framework 2004–2014*; www.hm-treasury.gov.uk/media/33A/Abspend04_sciencedoc_1_090704.pdf.
9. Moore, G. *Crossing the Chasm*. Harvard Business (1991).
10. Patterson, D. Restoring the popularity of computer science. *Commun. ACM 48*, 9 (Sept. 2005), 25–28.
11. Tew, A., Fowler, C. and Guzdial, M. Tracking an innovation in introductory CS education from a research university to a two-year college. In *Proceedings of ACM SIGCSE* (2005), 416–420.
12. Yaeger, T. Innovate or take a walk. *Infoworld* (Apr. 19, 2004), 69.

Peter J. Denning (pjd@nps.edu) is the director of the Cebrowski Institute for information and innovation and superiority at the Naval Postgraduate School in Monterey, CA, and is a past president of ACM.
Andrew McGettrick (Andrew.McGettrick@cis.strath.ac.uk) co-chairs the ACM Education Board and received the SIGCSE lifetime educator award in 2005.