

The most obvious issue facing computer science departments is low enrollments. Beyond how computer science departments will fare with fewer students and whether they can maintain a critical mass of faculty for intellectually viable programs “downsized” for lower enrollments, are questions about whether the U. S. will have people to maintain its already questionable high tech lead. Will we have people to fill critical jobs, conduct research, become teachers, understand the politics or sociology or business of the technology, etc?

Without research into why students are not coming to computer science, I can only speculate as to why we have low enrollments. Because I don't know where computer science cognoscenti will work, I can't argue convincingly for the curricula we should plan. The education we now offer will continue to serve some, since we will still need new architectures, compilers, programming languages, and operating systems. But, what will other computing professionals do? Armed with research results, I might argue that the computing our future graduates do will differ significantly from what they now do. I might argue for a minimal core – a concepts-based education, with skills (technology training) adequate only to deeply understand the core concepts and to learn how to acquire abilities such as problem solving and debugging and to gain new technical skills such as programming languages or environments. And I might suggest that many students should minor in a second field, and that all should do some advanced work and a significant project.

Because I don't know why so few undergraduates are majoring in computer science or how we might fix that, and because low enrollments seem to be outcomes of a cultural phenomenon that has little to do with computing per se, I will discuss instead the context in which enrollments rose and fell, touching on why I think we had perhaps abnormally high enrollments in the 1980's. I will then focus paper on how we might ameliorate the situation by teaching more computer science across the undergraduate curriculum.

We remain a new discipline in the academy, and our near-meteoric rise came about because of a popularity among students unlikely to recur. Yet, in spite of this early popularity, few colleagues, students, students' parents, or even collaborators in other disciplines, understand computer science. So, rather than suggest the computer science curriculum we should offer future majors, and how to assure a flow of freshmen knowing they want computer science, I will talk about ways to bring computer science to non-majors, and hope to convince at least some of you that enough of this will sustain and strengthen our field in important ways.

To explore how we became a discipline without really becoming part of the academy, I'll tell a story. In 1983 when I left industry to teach computer science, my state was experiencing structural unemployment in its traditional economic sectors, and new immigrants from South East Asia who valued education and the upward mobility it promised comprised 10-15% of our computer science undergraduates. Industry and business sought programmers for mainframe applications, and universities sought Ph.D. students to conduct research. Undergraduates flocked to computer science. In addition to a majority coming for a more secure middle class job than their parents had, was a significant minority of very bright students intrigued by a new dynamic discipline. Small colleges started up departments to meet this demand and to combat dropping enrollments elsewhere, but could not find faculty to staff new programs. R1 institutions complained that industry hired the recent Ph.D.s, “eating our seed corn”. NSF programs retrained Ph.D.'s from other fields in computer science. Enrollments continued to grow in the 80's and 90's, given the technology innovation of the PC, and even bad courses taught by uninspired faculty filled. We spent more energy keeping students out of our courses than welcoming or recruiting, and had little time to teach students in other fields how to use the technological fruits of our research or develop applications to solve meaningful problems in their own disciplines. Overwhelmed by the numbers of our own majors and the intrepid pace of technological change, we (with notable exceptions) left the job of educating non-majors about computing to other departments. And, we felt so overwhelmed by the large numbers of students that we tended to isolate ourselves from faculty in other disciplines.

Then, to quote Heller, “something happened”. Computing became a commodity, and application programming began to mean modifying turn-key systems. There came to be but two major operating systems and two platform alternatives in personal computing. The most interesting applications on the surface are computer games – which hold negative appeal for at least half of the populace. Today at my small state-supported liberal arts college the introductory sophomore-level computer science program that once enrolled full-time and year-long 60-75 culturally and ethnically diverse students and turned away another 40, barely makes an enrollment

of 35 that justifies the needed two teachers. Classes are often less culturally diverse than the faculty! Attrition rates are about the same as before, but hurt more because we start each year under-, not over-enrolled. At first embarrassed at these numbers, thinking that our program had lost its luster, on talking with other faculty we found their enrollments also down.

A small (even in our hey days) contingent in a sea of faculty from established disciplines, we fight to fill vacancies made by retirements because administrators want new positions in fields where student demand is high: business, health sciences, media arts, and ecology. Because we never before had to justify ourselves and because few of our colleagues really understand what we do and why we're critical to the academy, we don't always make a convincing case for computer science – or that we need a critical mass to teach major and non-major curricula, or that our institution must continue to graduate students who go into this field, or finally that we can't claim to “liberally and thoroughly” educate students if they don't understand computing and informatics concepts.

I propose we work towards: 1) ensconcing ourselves more integrally within our universities, and 2) developing curricula that assure an understanding of computing among significant numbers of students with primary interest in *other* fields. In two pages, I can't address both these, so will focus on the second idea. Even if the number of computer science majors soon increases as a result of herculean effort, future interest will likely wane again given cycles of student interest. Why not establish ways for students in other majors to understand why innovation in our discipline is relevant to them, and assure that they see what we do as intellectually engaging and important? I think we should further aim to engender in some non-computer science majors an understanding of and interest in our discipline sufficient for lateral moves into computing, whether at the undergraduate, professional, master's, or Ph.D. level.

As a computer scientist whose research and professional experience has centered on real-life problems in other areas, and ways that computer science abstractions interesting for their own sake can be relevant to a range of domain problems, I see the most critical of the world's problems as inherently interdisciplinary, many needing good computer science thinkers working with good thinkers from other disciplines. I see problems and opportunities for computing in media arts, ecology, linguistics, medicine and health care, molecular biology, physics, and theoretical chemistry. Students, academics, and practitioners from those fields see similar problems, but don't always see how computing innovation might help. To me, it is clear that some critical problems would be best addressed by an interdisciplinary team where one collaborating contingent bring to bear not merely the technology du jour (or of yesteryear), but modes of thought and problem solving methods unique to computer science. And some problems will inspire new computer science.

What if half or a fourth of freshmen students took a computing course that taught fundamental concepts – the great ideas in computer science? What if many non-CS disciplinary preparations included the ways computing is currently used and why it is critical – perhaps in courses team-taught by a domain expert and a computer scientist? What if all students pursued senior theses or capstone projects, and presented their work at undergraduate colloquia? What if fields where innovations in computing could make a significant difference encouraged students to pursue senior theses or projects under the direction of computer science faculty whose own research or professional expertise touched that discipline? What if we computer scientists opened our own capstone project courses to students from other disciplines, and made it easy for those students to prepare for such work by taking the equivalent of a minor in computer science? What if we computer scientists sought collaborators among innovators in other fields, and encouraged our own students to work jointly with students from other fields on capstone projects? What if we computer scientists began working for curricular and research innovation outside, as well as within, our field?

Finally, I ask that we retain in computer science a sense of its inter-disciplinarity and of the relevance of other fields to our own. In addition to growing the number and building the diversity of students who pursue computing, I fully believe that such outreach to and synergy with other disciplines will lead to interesting innovations within computer science. Thus, I propose that NSF fund curricular experiments that cross disciplines, and work with faculty across the board to bring them fully into the academy.