

Challenges in Computing Education

Tim Sheard, January 2006

I have two issues I'd like to see the meeting address.

- **Well Designed Feeder Programs into Computer Science.** If the country needs more and better prepared computer science graduates, they have to come from somewhere. I think we should address where these students will come from. We can't just assume that by attracting many new students into higher education, Computer Science will get its fair share of the increase. I believe we need to redirect students from other disciplines into Computer Science. I believe students choose majors for many reasons, not all of them rational or well-thought out.

One ill-thought out reason is the perception that some majors lead to careers that pay especially well. This was one of the driving forces behind the large increase in C.S. majors in the 80's and 90's, and behind the recent rise in enrollments in many engineering disciplines. Of course the fallacy in this argument is that the hot majors come and go, and its unlikely that one's choice in major will be the high paying major for a lifetime.

A second reason for choosing a major is early exposure to an area. This reason is quite rational from the students perspective, but also quite unfortunate for Computer Science as a discipline. Computer Science in secondary education is simply missing from the curriculum, and early exposure to Computer Science (not computers) is rare.

In secondary education, all college bound students take Biology, Chemistry, Physics, and several years of Mathematics. Students are exposed to computers, but not the science of computing. If they are exposed to computing, it is to decade old ideas by poorly prepared teachers. In my own state of Oregon, every secondary teacher must be certified in some specialty. There are dozens of specialties. Computer Science is not amongst them. So any additional training to teach Computer Science in high school comes above and beyond the required certification. There are strong disincentives to becoming an effective teacher of Computer Science!

We need to do something about this. Colleges should demand well designed instruction in Computer Science at the secondary level. As professors of Computer Science, we need to get involved suggesting curriculum and by training and mentoring our secondary education colleagues. We need direct outreach to high school students through enrichment programs and opportunities for high school students to see Computer Science at work.

As a society, I think we should rethink some of the standard curriculum at the secondary level. Why do we teach Geometry at the secondary level? Perhaps, because it is a student's first introduction to formal proof and mathematical thought processes. Is it the Geometry that is important? or the thought processes? Couldn't we teach the same thought processes using the finite mathematics that underlies much of Computer Science?

- **Teaching Programming Effectively.** We all agree that Computer Science is more than programming. But I am amazed at how rare it is to find a good programmer amongst our students and our graduates. This seems to be true at all levels: I see this trend in undergraduates and graduate students, even in non-matriculated "curiosity driven" students based in industry. I find that students have difficulty in understanding advanced programming techniques, and cannot utilize them in their programs. This hinders our students in many ways.

I think we need to approach the teaching of programming they way we approach the teaching of writing. We teach writing by practice, study, and critique. Most colleges have required writing programs where students complete many writing exercises. These exercises are critiqued by instructors who comment not only on the effectiveness of the writing, but also on its style. Students are required to study the writing of others. The study of classic literature is an end in itself. Imagine spending an hour or more critiqueing each student programming exercise. Imagine studying the GNU compiler for its elegance and beauty. We don't spend the time or resources we need to develop really good programmers, and it shows.

I believe there are strong parallels, that we cannot safely ignore without putting the success of our discipline in grave peril.