

## Stanford ICER White Paper

In this paper I would like to address two areas that represent, more or less, the content and the delivery of computing education.

We are all aware of the broadening incursion of computing into all other aspects of life, and the integration of computing with many other applications. Yet our students typically begin their computing education with a very low-level "hello world" programming approach in some language or other. Even a very object-oriented or visual or functional approach is still about programming solutions to a series of small and usually contrived problems that the students know don't have much impact. At the other end of the student's 4-year education plan, there is often a software engineering course, where the projects are larger, students may interact with a real customer, and topics of project management, testing, budget, the importance of specification, etc., come into play. In between, students get exposed to data structures, architecture, networks, programming language theory, formal languages, database, discrete math, or whatever else we think is important. In this traditional curriculum model the student begins by working at a close-to-the-machine level (even if the first course is no longer, for example, assembly language), and ends up interacting, often as part of a team, with some real-world customer or problem.

I would like to see this model turned inside out so that students begin with exposure to what I will call software engineering and gradually fill in the technical underpinnings as needed. This would be an approach similar to medical school education. Medical students follow practitioners around in clinics, listen and learn about various "real world" cases, and gradually, as their education progresses, assume more responsibility. Even the formal part of medical school education now is somewhat "case-based".<sup>1</sup>

Suppose the entire undergraduate department operated a public software engineering clinic, as a medical school teaching hospital operates a dermatology clinic, a pediatrics clinic, etc. Clients submit projects to the software clinic (real but not critical-path projects). Freshmen participate at the levels of client interview observations, checking requirements vs. specifications, software testing, budget and progress monitoring. These are relatively soft skills that require little technical training. Sophomores, juniors, seniors participate with more and more responsibility. Along the way, technical skills have to be taught in such a way that students perceive them to be a "need-to-know" motivated by present or past

---

<sup>1</sup> [http://www.usc.edu/schools/medicine/education/degrees\\_programs/mdp/md/new\\_curr.html](http://www.usc.edu/schools/medicine/education/degrees_programs/mdp/md/new_curr.html)

projects. Some amount of application area knowledge should be included as well - students need to learn a little bioinformatics or something about the electric power industry, or whatever the projects require. Security issues, so all-pervasive, need to be worked into the mix relative to the current project(s).

What are the advantages to such an approach? Students see how computing arises in many different applications, they have a sense of real-world participation and, one hopes, motivation. They early on learn the soft skills that potential employers keep telling us they want - teamwork and communication skills. Employers get some back-burner project work done and have the chance to select good future employees, much as student internships give them. Projects for non-profit agencies could strengthen University/community relationships.

What are the requirements for such an approach to come to pass? For the faculty, a willingness to redesign the entire curriculum and a great deal of management and knowledge integration to be sure that students do graduate with a solid technical background. For the university administration, a willingness to allow degree programs to be designed without the usual so-many-3-credit-boxes/semester approach, while still being assured that educational standards are met. For the professional accrediting agencies, the same flexibility. For industry, the willingness to be full partners and participants in this approach - they must be the clinic's "patients." Whereas once this would have been impossible in rural areas or small schools with little industry or business nearby, it can now certainly be done via electronic conferencing and other electronic communications media. For government, funding that encourages development of new curricula designed with this radical approach, faculty training programs (much like the old retooling for CS programs in the 1980s), and facilitation of industry/university contacts and communication.

My second comment concerns modes of delivery. As a computer science educator, I find the biggest challenges today to be a declining ability or inclination of students to read and a declining mathematical ability. Technical content could be delivered in a total package using some form of portable electronic book mechanism that can contain all text material, audio files, video clips, a mathematical tool, all necessary compilers and other software, of course all hyperlinked and searchable as needed. This Google-like delivery may be more effective with today's students than traditional textbooks and lectures.