

ICER White Paper

Preparing Students for Careers in 2015^{**}

*Lawrence Snyder
University of Washington*

This white paper is organized as bullet points since there is insufficient space to fully elaborate the main ideas.

Binary Focus: Though accommodating the educational needs of our rapidly changing field will take a variety of improvements, two seem most critical to me:

- More Independent learning
- More “Experimental” learning

An Explosion of Specialties/Information: The explosion of CS-related specialty fields causes the “common core” of knowledge shared by all practitioners to shrink. Further the dramatic expansion of information in each field due to such features as increased size and velocity means each specialist must learn more. These two facts suggest that a student’s educational needs will likely mismatch the standard “coarse grain” courses offered in contemporary curricula. A principles-based education assures that each graduate is well able to learn a specialty.

More Independent Learning: A solution is to teach foundational material, which most programs are already doing to some degree, and then rely on independent learning by the student to fill in the details just in time. (“Foundational” here doesn’t refer to “theory,” but rather to the underlying ideas of a topic area, which might include theory as it does in compilers.) But self-directed learning is not widely taught, and many students in my experience are ineffective at it. A Foundations+JIT Learning approach works well in CS-related fields, because the specific knowledge needed to achieve a particular goal often has little value in other situations, and therefore isn’t worth knowing except at the moment it’s needed. The technique addresses the “explosion of information” problem cited above.

Of course, all education is supposedly directed towards creating independent learners, but the proposal here is to organize a CS education (and therefore the curriculum) around that principle explicitly. The most compelling justification for the approach is that it aligns so well with the field. It is in the nature of computer science as an intellectual discipline to work out the nitty-gritty details (the program, design, layout, etc.) for large and often complex abstractions (the algorithm, web page, chip, etc.). This is analogous to the Foundations + JIT learning. Self directed learning must become an item in the

^{**} With all due respect to whoever set the 2010 target and with no intent to offend, substantive evolutionary change cannot happen so fast, and revolutionary change is not appropriate for education. All of the students who will get BA/BS/BSE degrees in CS-related areas in 2010 *have already applied to college* and some are already admitted. A 2015 goal is the most ambitious possible.

curriculum, and the foundations courses need to be expanded to cover the foundations (only) of more disciplines.

Exploring the Unknown. With the expansion of the field and the explosion of knowledge, computer science is the antithesis of traditional fields like math and physics with their well-studied, compartmentalized content. CS is being created on the fly. Accordingly, many “subjects” will be little studied and graduates will be obligated to extend what is known with their own research. This will generally be “experimental” research, where that term is used as in the NRC report [1]. That is, it will entail design and implementation of systems—generally software—as well as its testing and evaluation.

Experimental Computer Science. The curricular need motivated by an expansion of experimentation is first to teach the methodologies explicitly, and secondly to give students ample opportunity to practice the methodologies. Capstone design classes in engineering programs fill this need at present, but more is needed. Project based classes and annual opportunities to take capstone-like courses are ways to increase the opportunities for students to encounter design.

Requirements. The two recommendations operate within the existing college structure, but they both require change “in the way we do things.” First the foundations classes must be established, but on a broader footing than the traditional curriculum. So, for example, bread-and-butter courses like compilers, programming languages and software engineering that overlap in their content would not be taught as separate classes as they are today, but rather they would be replaced by a foundations course—Foundations of Programming Systems, perhaps—that introduces the (common) underlying ideas and tools; further the class would explain the specific concerns of the constituent specialties. Such a course doesn’t exist today; a syllabus, textbook, assignments, etc. must be developed. Other areas would be similarly grouped. The “independent learning” and “experimental methodologies” courses need to be created as well.

Second, the concepts described in the foundation classes must be applied in practical situations in “experimental” classes. The goals would be (a) to remove artificial barriers as they exist among the traditional topics, (b) attempt more substantive projects, (c) learn the techniques of JIT learning, and (d) apply the methodologies of experimentation. So, continuing the example above, Foundations of Programming Systems might build an IDE in a design class rather than making a toy compiler as part of a compiler class.

Summary. We have discussed two ways to revise the undergraduate computer science curriculum so that it fits within existing college structures while addressing pressing problems in the field. The first involves focus on the “foundations or principles” classes coupled with just-in-time learning, and the second involves focus on experimentation. Together, they support the view that teaching more is hopeless, but that teaching the skills of learning independently make learners both adaptive and effective.

1. NRC, Academic Careers for Experimental Computer Scientists and Engineers, NAP, 1992.