EMERGING THEMES IN COMPUTING CURRICULA FOR SCIENCE AND ENGINEERING

Allen B. Tucker,
Bowdoin College
e-mail: allen@polar.bowdoin.edu

I. Introduction

During the last decade, dramatic changes have taken place in the definition and teaching of the discipline of computer science and engineering. The recent decline of U.S. undergraduate enrollments, from their peak of 42,100 in 1986 to a low of 30,900 in 1989 [2], has been attributed to two major causes: saturation of the employment market and lack of clarity in the definition (and the undergraduate curriculum) of the discipline of computer science and engineering itself.

In response to the second problem, several efforts were initiated in the late 1980s and early 1990s. These efforts were aimed at developing a comprehensive definition of the discipline of computing, designing a new model curriculum for undergraduate major programs, and class testing various aspects of that model in a wide range of actual undergraduate programs in the U.S.

This paper summarizes the results of these efforts. In section II, we review the major recommendations for curriculum reform that appear in the Denning Report [3] and Curriculum 91 [1]. In section III, we review the more recent recommendations that arise from the High Performance Computing and Communications Act [5] and the report Computing the Future [2]. In section IV we introduce the design of the Rbreadth first curriculum, which is a model introductory curriculum aimed at implementing these recommendations. There, we also summarize the results of class testing that curriculum in several colleges and universities. Section V concludes this paper with some thoughts on future computing curricular needs for science and engineering.

II. Redefining the Discipline and the Undergraduate Curriculum

In 1989, an ACM-commissioned task force issued a report which has since become known as the "Denning Report" [3]. Its purpose was to propose a new, comprehensive definition of the discipline of computing (read computer science and engineering) and then to suggest an alternative paradigm for teaching the introductory courses in the undergraduate curriculum in computing. The definition of computing was presented as a two-dimensional matrix, with nine different subject areas identified in one dimension and three different paradigms identified in the other. The subject areas were:

- Algorithms and Data Structures
- Architecture
- Artificial Intelligence
- Database and Information Retrieval
- Human-Computer Interaction
- Numerical & Symbolic Computation
- Operating Systems
- Programming Languages
- Software Engineering
The three paradigms were identified in order to acknowledge that computer scientists and engineers view each of these subject areas from different perspectives, essentially that of a mathematician, that of a scientist, and that of an engineer. A fourth paradigm was added in the subsequent Curriculum 91 report [1], which acknowledged computing uniquely and directly affects the quality of peoples' lives and thus reflects the perspective of the social scientist. Thus, the four paradigms:

- Theory; including mathematics, logic, and formal methods
- Abstraction; including scientific inquiry, modeling, and hypothesis testing
- Design; engineering methods, the life-cycle design model, and cost/benefit analysis of alternatives
- Social and Professional Context; including the computer-human interface, liability for errors, intellectual property issues

Based on these paradigms, the Curriculum 91 report outlined an approach to curriculum design that would ensure that students gained significant exposure to all nine subject areas and all four paradigms during the course of an undergraduate major. This report also recognized that different types of programs would naturally implement these ideals through different sets of courses. Hence, an undergraduate curriculum modeled after the Curriculum 91 recommendations would have the following characteristics:

- Breadth of discipline coverage
- Integration of theory, abstraction, design
- An improved laboratory methodology
- Coverage of social and professional issues

III. Developments Aside from Curriculum 91

Independent from the development of Curriculum 91, the US Congress passed legislation called the "High Performance Computing and Communications Program" [5]. This program, called HPCC, encouraged universities, research programs, and industry to develop:

- High performance computing systems: to improve computing speeds by 2-3 orders of magnitude
- Advanced software technology and algorithms: to support the "grand challenges" in science and engineering
- Networking: to support R&D for a gigabit National Research and Educational Network (NREN)
- Human resources: expanding basic research in all areas of computing that are relevant to high performance computing

The "Grand Challenges" that were identified in HPCC are those fundamental problems in science and engineering with potentially broad economic, political, or scientific impact that can be advanced by applying high performance computing technology. Moreover, these problems are achievable only by high-level collaboration among computer professionals, scientists, and engineers. These problems include: the prediction of weather, climate, and global change; semiconductor design; drug design; the human genome project; quantum chromodynamics; naval architecture; quantum chromodynamics; astronomy; structural biology; superconductivity; underwater acoustics; modeling; vision.

A second development since the publication of Curriculum 91 is published in the report Computing the Future (CTF) in 1992 [2]. This report addressed the need for the
computing field to broaden its research agenda and its educational horizons, so that computer professionals would be better prepared to help meet the grand challenges outlined in the HPPC. Specifically, the research agenda should be expanded to include: computational science; commercial computing (MIS, Organizational Informatics); automated libraries; environmental modeling. Educational horizons should be expanded to strengthen university-industry ties, undergraduate programs, and undergraduate service education.

Specifically, CTF recommended that undergraduate programs need to add principles and rigor in their software design courses; stress mathematics and formalism; and improve students' breadth of understanding, both among the paradigms of theory, design, and applications and among functional, logic, object-oriented, and parallel programming paradigms.

CTF also emphasized the importance of undergraduate programs teaching the social implications of technology to nonmajors and majors, actively promoting minors and double majors, and teaching the computing skills necessary for students majoring in related disciplines (e.g., physics, chemistry, biology, economics, and engineering).

These three efforts, Curriculum 91, HPCC, and CTF, are clearly complementary in their recommendations for undergraduate curriculum reform. Since Curriculum 91 was published, many efforts have been made to redesign traditional curricula so that these various goals will be better addressed. One such effort is the design and implementation of the "breadth-first curriculum" [11] that was originally suggested in the Denning Report and refined in Curriculum 91.

IV. Design of the Breadth-First Curriculum

The goals of the breadth first curriculum embrace those set forth in Curriculum 91 itself:

- To introduce the richness (breadth) of the discipline, beginning with the first course
- To provide an integrated and well-structured laboratory component for each course

The four key courses that form the foundation of the breadth first curriculum are called:

Course I: Logic, Problem Solving, Programs, and Computers
Course II: Abstraction, Data Structures, and Large Software Systems
Course III: Levels of Architecture, Languages, and Applications
Course IV: Algorithms, Concurrency, and the Limits of Computation

Course I (which is a broadening of the traditional CS1 course) is the first course that students take. That course may be directly followed by Courses II (a broadening of the traditional data structures course) and III (a broadening of the traditional computer organization course). Course II is a prerequisite for course IV (a broadening of the first algorithms course). The content of these courses is summarized as follows:

Course I: a study of logic, programming, fundamentals of computer architecture, and copyright issues
Course II: an introduction to object-oriented methodology, data structures, operating systems, and software liability
Course III: a study of logic design, computer organization, AI principles, and Lisp machines (interpreters)
Course IV: a study of graph algorithms, hard problems, parallel and heuristic strategies, and unsolvable problems

Each course is accompanied by a weekly scheduled laboratory experience, which serves to introduce students to the methodology of computer science and engineering and reinforce principles taught in the lectures. The breakdown of weekly laboratory work is summarized roughly as follows:

Course I: programming (9 weeks)
          computer organization (4 weeks)
          social issue (1-2 weeks)
Course II: objects and classes (3 weeks)
          data structures (8 weeks)
          operating system simulation (3 weeks)
Course III: logic design (5 weeks)
           assembly language (5 weeks)
           Lisp machine architecture (4 weeks)
Course IV: graph algorithms (6 weeks)
          parallel algorithms (4 weeks)
          Universal Turing machine, halting problem (4 weeks)

Each weekly laboratory assignment is designed in a way reminiscent of a traditional science lab experiment. That is, students are given a worksheet that describes the goals of the lab, summarizes the necessary technical background, suggests a methodology of approach, and asks a few questions about the outcome of the exercise. While students begin the lab in a single group (20 or fewer students is ideal), some labs will require further work at a later time. Typically, laboratory assignments are due at the end of the week in which they are begun. At that time, students hand in their completed program, output, other results, and their written answers to the questions.

The teaching and laboratory materials for courses I and II in the breadth-first curriculum have been published as textbooks, laboratory manuals, and software sets [9,10]. These materials have allowed the courses to be taught in a wide variety of institutional settings, including colleges of liberal arts, sciences, and engineering. We have class-tested one or more of these four courses (or their variants) at each of the following institutions:

- Allegheny College (courses I and II)
- Bowdoin College (courses I, II, III, and IV)
- University of Connecticut (courses I, II, and III)
- Swarthmore College (course II)
- University of Texas at El Paso (course I)

In this class-testing, we sought student and instructor opinions on the effectiveness of each course's topic selection (breadth) and laboratories. The results of this class-testing are summarized below.

On the positive side, students and instructors were particularly enthusiastic about the quality and effectiveness of the laboratory experiences throughout courses I, II, III, and IV. Students voiced a strong preference for retaining a strong programming component in course I; that is not turning course I completely into a survey course.
Students also strongly applauded the introduction of nontraditional subject matter in these courses; notably:

- Computer organization and social issues in course I
- Object-oriented design and operating systems in course II
- Lisp (AI) and logic design in course III
- Parallel programming and hard problems in course IV

On the less positive side, students voiced dislike for the following topics:

- Proof and program verification in Course I
- Complexity and sorting in Course II
- Computer organization in Course III
- Unsolvability and the halting problem in Course IV

This is somewhat disconcerting, since many of these topics are traditional mainstays of the introductory computer science curriculum.

An overall conclusion that we draw from this class-testing is that the breadth-first curriculum is viewed as a refreshing alternative to the traditional curriculum, for students and instructors alike. However, as these courses mature and evolve, adjustments should always be made to tailor them to the needs of individual curricula and students. A more extensive paper [8] gives further details about this entire development and class-testing process for the breadth-first curriculum.

V. Emerging Themes and Conclusions

During the class-testing of the breadth-first curriculum, and as a result of HPCC and CTF, the discipline of computer science and engineering has continued to evolve and expand. Two entirely new subject areas, called Computational Science [6] and Organizational Informatics [4] have emerged to alter the framework of the original nine subjects identified in the Denning Report.

At the present time, this author is heading a project to develop the first edition of the "CRC Handbook of Computer Science and Engineering" [7]. This Handbook, which will have 2500 pages and over 200 contributing authors, will provide a comprehensive reference for practitioners and researchers in the discipline of computing, as well as other scientists, engineers, and information specialists whose work overlaps or collaborates with computing in a substantive way. The design of this Handbook is based on the nine subject areas of the Denning Report and Curriculum 91, as modified by the emerging subject areas that are identified in HPCC and CTF. That is, the Handbook's outline now reflects the evolution of the discipline of computing through 1995 into the following ten subject areas:

- Algorithms and Data Structures
- Architecture
- Artificial Intelligence
- Computational Science (including numerical and symbolic computation)
- Database and Information Retrieval
- Graphics (distinct from Human-Computer Interaction)
- Human-Computer Interaction (including organizational informatics)
- Operating Systems and Networks
- Programming Languages
Software Engineering

Some of the distinctions revealed by this new list of subject areas for the discipline are obvious, while others may be less clear. For instance, computer graphics is a separate subject area here, while it was originally embedded as a sub-area of HCI.

Computational science [6] has evolved out of the original area called numerical and symbolic computation, and now encompasses the fundamental connections between computation and scientific research. For instance, fields like computational astrophysics, computational fluid dynamics, and computational chemistry all emphasize applications of computing in science and engineering, algorithms, and in many cases special considerations for computer architecture. The emergence of computational science was indeed motivated by the grand challenges of the HPCC [5].

Human-computer interaction is also a field that has evolved rapidly since in the last few years. It now includes topics comprise the emerging sub-area of organizational informatics [4]. Organizational informatics also was motivated by the grand challenges of the HPCC. It emphasizes applications of computing in business and management, information systems and networks, as well as their implementation, risks, and human factors.

Another need that was identified by the HPCC and developed in CTF [2] is that of developing new university-industry ties in undergraduate and professional education. This includes the possibility of cooperation in curriculum development and teaching, collaborative learning experiences, and sharing technical and educational resources (especially teachers and professionals). The CRC Handbook of Computer Science and Engineering is a testimony to the feasibility of such collaboration, since many of its chapters are authored by industry professionals while others are authored by college and university researchers and educators. Other pilot projects that are aimed at developing collaboration are beginning to emerge. For instance, an NSF-supported project aims to develop a collaborative between the University of Washington and Boeing to develop and teach a computing curriculum to professionals and undergraduates, using university faculty and Boeing technical staff working in unison.

In conclusion, we see that new trends in undergraduate computer science and engineering education are emerging rapidly. Curriculum changes are including new subject areas at all levels, beginning with the first course, and with some success. Curriculum changes are also developing an improved laboratory methodology and a broader sense of the discipline among undergraduates. In the future, curricula should continue to evolve to take into account new subject areas suggested by computing applications in science and industry. They should also prepare students better to enter the workforce and contribute to technology in a wide range of scientific and engineering fields. Teaching methods should concurrently evolve away from the pure "lecture" method, and emphasize student collaborative work at all levels. Finally, support and rewards should be built more favorably into the promotion and tenure scheme, to provide interested faculty a stronger incentive to develop new curricula that will continue to satisfy emerging needs.

References