

Technical Report
CMU/SEI-94-TR-11
ESC-TR-94-011

1



Carnegie Mellon University
Software Engineering Institute

AD-A280 940

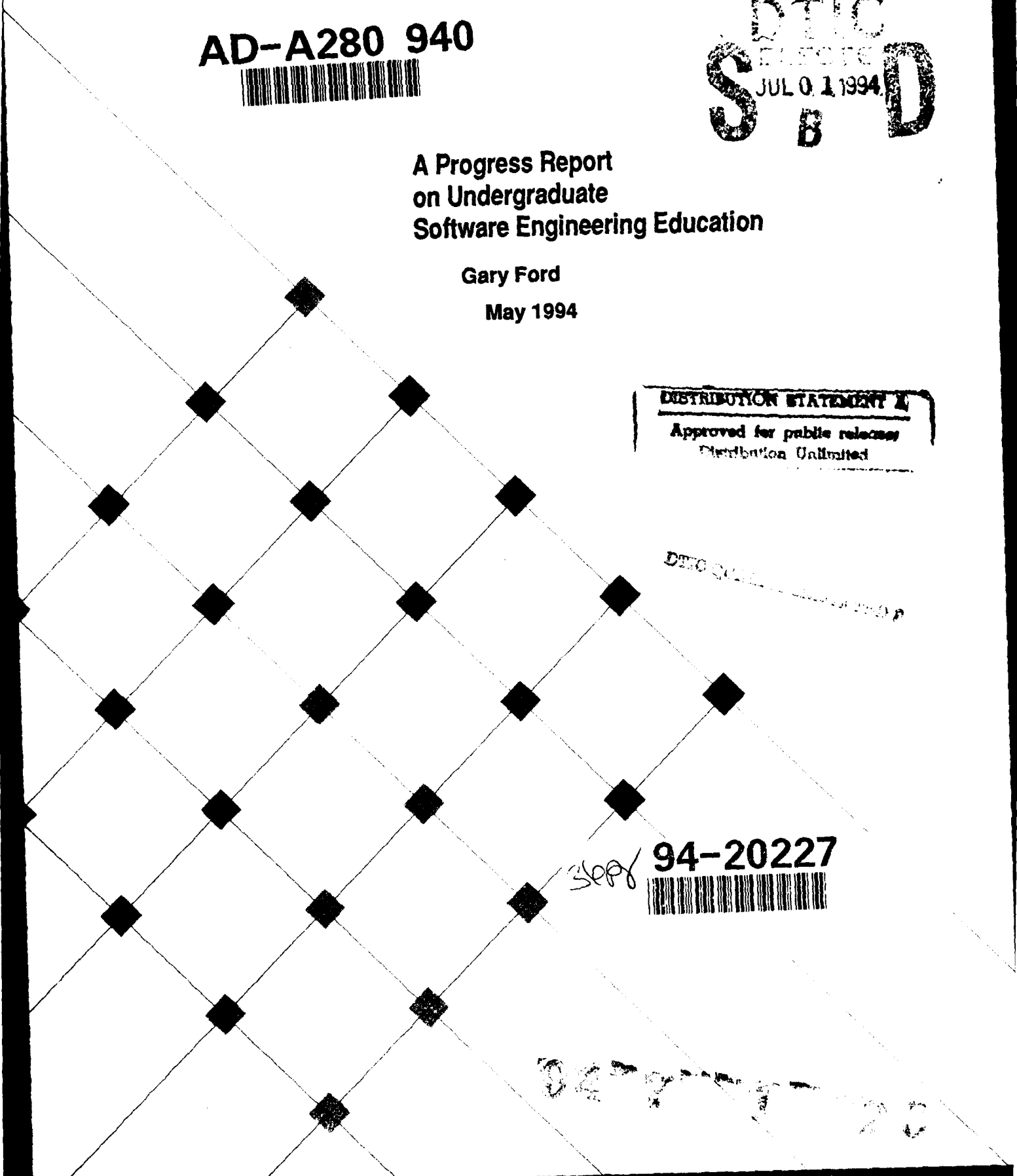


DTIC
S B D
JUL 0 1 1994

A Progress Report
on Undergraduate
Software Engineering Education

Gary Ford
May 1994

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited



DTIC QUALITY INSURED

SEP 94-20227



94-20227

Carnegie Mellon University does not discriminate in its educational programs on the basis of race, sex, religion, national origin, or ancestry. In addition, the University does not discriminate in its educational programs on the basis of age, marital status, or disability. The University's policies are based on the University's Charter, the University's Bylaws, and the University's Board of Trustees' Amendments of 1979 and 2001 and 2011. The University's policies are also based on the University's policies on non-discrimination.

In addition, Carnegie Mellon University does not discriminate in its educational programs on the basis of sex, race, national origin, or ancestry. In addition, Carnegie Mellon University does not discriminate in its educational programs on the basis of age, marital status, or disability. The University's policies are based on the University's Charter, the University's Bylaws, and the University's Board of Trustees' Amendments of 1979 and 2001 and 2011. The University's policies are also based on the University's policies on non-discrimination.

Inquiries concerning special needs of students should be directed to the Office of Disability Services, Carnegie Mellon University, 412-268-2056. The Office of Disability Services is located at 412-268-2056.

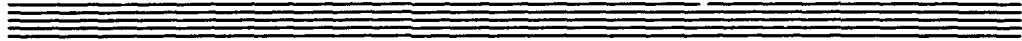
Technical Report

CMU/SEI-94-TR-11

ESC-TR-94-011

May 1994

A Progress Report on Undergraduate Software Engineering Education



Gary Ford

Curriculum Research Project

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

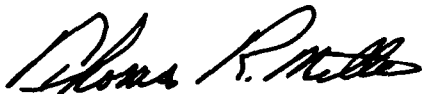
SEI Joint Program Office
HQ ESC/ENS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Copyright © 1994 by Carnegie Mellon University

This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994.

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145. Phone: (703) 274-7633.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

1. Introduction	1
2. The Current Status of Undergraduate Software Engineering Education	3
2.1. Oregon Institute of Technology	3
2.2. Parks College of St. Louis University	4
2.3. University of Detroit Mercy	5
2.4. Rose-Hulman Institute of Technology	6
2.5. Drexel University	6
2.6. State University of New York, Oswego	7
2.7. University of Virginia	8
2.8. University of West Florida	8
2.9. Rochester Institute of Technology	8
2.10. University of Washington	9
2.11. Florida Institute of Technology	10
2.12. Other Schools	11
3. Evolution of Undergraduate Software Engineering Programs	12
3.1. Growth of Graduate Software Engineering Programs	12
3.2. Evolution of Computer Science Programs	14
3.3. The Computer Engineering Program at Carnegie Mellon University	19
3.4. Conclusions	23
4. Recent Actions by the Professional Societies	25
4.1. IEEE Computer Society	25
4.2. ACM	27
4.3. Task Force Activities	27
References	28

Accession For	
DTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution/_____	
Availability Codes	
Dist	Avail and/or Special
A-1	

Table of Figures

Figure 2.1. Locations of Undergraduate Programs Surveyed	3
Figure 3.1. Locations of Software Engineering Graduate Programs	14
Figure 3.2 Growth of Master's Programs in Software Engineering	15
Figure 3.3a. CMU ECE Curriculum: Freshman and Sophomore Years	20
Figure 3.3b. CMU ECE Curriculum: Junior and Senior Years	21
Figure 3.4 CMU Combined Electrical and Computer Engineering Curriculum	22

Table of Tables

Table 3.1. Graduate Programs in Software Engineering	13
Table 3.2. Graduate Programs in Computer Science with a Software Engineering Option	13

A Progress Report on Undergraduate Software Engineering Education

Abstract: The current status of undergraduate software engineering education in United States universities is summarized, including descriptions of programs at eleven schools. Possible scenarios for the further evolution of undergraduate software engineering programs are described, based on observations of the evolution of computer science and computer engineering programs. Recent and ongoing activities of the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM) regarding the establishment of the profession of software engineering are described, including the expected implications for undergraduate software engineering education.

1. Introduction

Software engineering education in United States universities is evolving rapidly. In the 1980s, master's-level software engineering programs were just beginning; there were few programs and their content varied considerably. The idea of a separate undergraduate software engineering program was so controversial as to be almost unimaginable. In the 1990s, master's programs are becoming more common and their content better defined. Many computer science doctoral programs have a substantial number of students conducting research and writing dissertations on software engineering topics. The idea of an undergraduate software engineering program separate from computer science, although still controversial, is being taken seriously by many people, and the first programs are being designed. By the end of the decade, we expect that software engineering degree programs at all academic levels will be well established.

Since 1985, the Software Engineering Institute¹ has tracked and reported on the growth of software engineering education. Previous reports [Ardis89, Ford91] have concentrated on graduate-level education. This report focuses on undergraduate education.

In Chapter 2, we describe programs in eleven United States universities that illustrate the range of approaches to software engineering education that are being tried.

¹ The Software Engineering Institute (SEI) was established at Carnegie Mellon University in December 1984, under a contract with the United States Department of Defense. Its primary mission is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software.

Chapter 3 compares the evolution and growth of software engineering with those of computer science and computer engineering programs. Some parallels are identified that suggest the future growth of software engineering programs.

Chapter 4 describes recent and ongoing activities of the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE-CS) and the Association for Computing Machinery (ACM). The two professional societies are cooperating on a range of activities that support the establishing of software engineering as a profession. Some of these activities will have significant implications for undergraduate software engineering education.

2. The Current Status of Undergraduate Software Engineering Education

We do not know of any undergraduate programs named “bachelor of science in software engineering” at any United States universities. However, several schools teach significant course sequences in software engineering at the undergraduate level, a few schools offer software-related programs (other than computer science), and some schools report that they are now developing undergraduate programs in software engineering. In the next sections we describe the efforts at eleven of these schools. Figure 2.1 shows the geographic distribution of the schools.

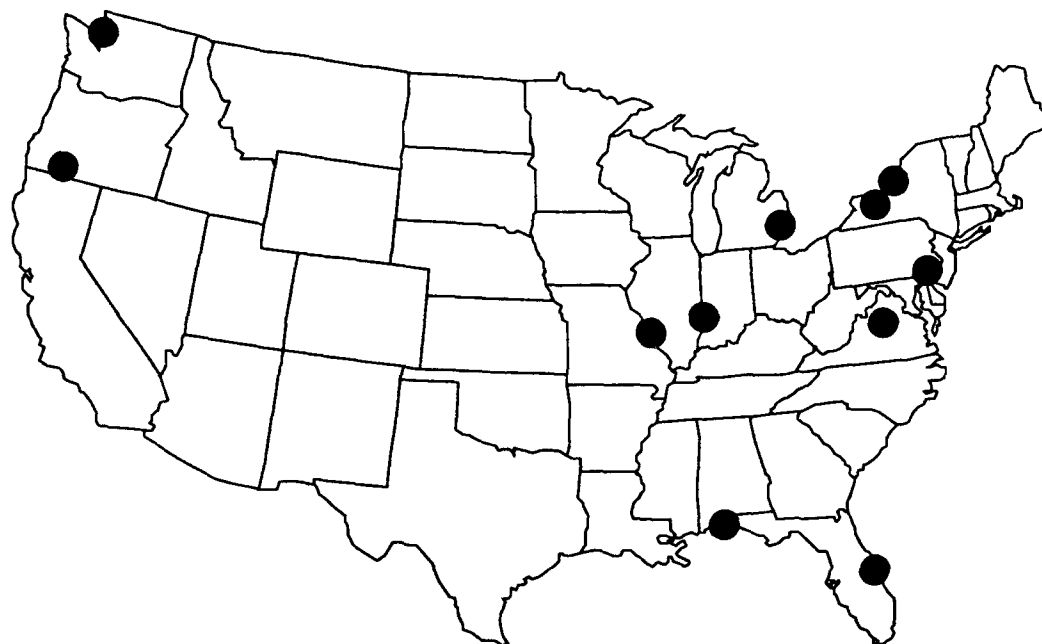


Figure 2.1. Locations of Undergraduate Programs Surveyed

2.1. Oregon Institute of Technology

The Oregon Institute of Technology (OIT), founded in 1947, is the polytechnic institute in the Oregon state university system. For several years it has offered a four-year program leading to the degree “Bachelor of Science in Software Engineering Technology” and a two-year program leading to the degree “Associate of Engineering in Software Engineering Technology.” Both programs were accredited by the Technology

Accreditation Commission of the Accreditation Board for Engineering and Technology (ABET) in 1991.

The OIT program announcement characterizes the programs in this way: "The study of Computer Science prepares individuals to discover and develop new ideas in the theory of computers and computing. Software Engineering Technology uses these ideas, together with sound engineering principles, to design and implement economical, reliable and maintainable software of all types."

The program is further described as using "... an applied approach to teaching. ... Many of these classes have a required laboratory component where the student learns to use state of the art tools such as: code development and test work benches, interactive debuggers & tracers, test generators, cross-compilers and emulators, as well as other Computer Aided Software Engineering (CASE) tools."

In addition to a range of courses in mathematics, science, humanities, and social sciences, the typical curriculum includes these technical courses:

Freshman: Computer Science I, II; C Programming; Advanced C Programming; Computer Architecture I, II; Computer Assembly Language

Sophomore: Advanced Assembly Language Programming; Programming Languages II, III; Data Structures; UNIX; Computer Graphics I; Data Base I; Introduction to Grammars

Junior: Computer Logic I, II, III; Compiler Methods; Software Design and Implementation I, II; Numerical Methods; Operating Systems

Senior: Computer Networks; Management Processes I, II; Senior Development Project; Industrial Psychology; Industrial Economics; Introduction to Artificial Intelligence

Note that OIT uses a quarter system, so it is typical for a student to take five courses in each of three quarters during the academic year.

For more information, contact: Oregon Institute of Technology, Computer Systems Engineering
Technology Department, Klamath Falls, Oregon 97601.

2.2. Parks College of St. Louis University

Parks College of St. Louis University specializes in aeronautics and flight-related education. Beginning in 1989, it offered a bachelor's degree in "Airway Science/Computer Science." In 1992, the curriculum was revised to provide two additional degree options, "Applied Computer Science" and "Computer Software Systems." It was the intent of the faculty for this latter degree to prepare students for a software engineering career, but for several reasons it was not possible to use the phrase "software engineering" in the title.

All three degree programs share a common freshman and sophomore year curriculum. In the junior and senior years, the applied computer science curriculum requires the students to choose a specialization in aerospace or electrical engineering, avionics,

meteorology, logistics, or aeronautical administration. In those years the Computer Software Systems students take several additional software-related courses, including

- Operating Systems
- Numerical Methods
- Introduction to Computer Software Systems
- Software Specification
- Software Design
- Software Generation and Maintenance
- Software Verification
- Software Design Project
- Software Project Management and Economics
- Current Topics in Computer Software Systems.

Two additional software-related electives are required.

For more information, contact: Parks College, St. Louis University, Department of Science and Mathematics, Cahokia, Illinois 62206

2.3. University of Detroit Mercy

Mercy College of Detroit recently merged with the University of Detroit, at which time the campus adopted the name University of Detroit Mercy. Both schools have offered undergraduate computer science programs for many years.

The College of Business and Administration now offers the degree "Bachelor of Science in Software Production and Management." The curriculum integrates recent software engineering knowledge with a more traditional curriculum in computer information systems. Its goal is to produce graduates who are competent at producing quality software products [Jovanovic92].

Core courses in the program are

- Business Programming
- Systems Analysis and Design
- Database Design
- Data Structures
- Specification and Design
- Software Quality Assurance and Testing
- Software Management
- Data Communications.

Electives include

- Interface Design
- *Advanced Project in Software Production and Management*
- International Software Management.

The school is also developing a graduate program, "Master of Science in Software Management."

For more information, contact: University of Detroit Mercy, Computer Information Systems Department, Detroit, Michigan 48221.

2.4. Rose-Hulman Institute of Technology

Rose-Hulman Institute is a small (1400 students) men's technological institution. In 1987, the school created an independent computer science department and completely revised the computer science curriculum. Before that time, computer science was taught in the electrical engineering department, and the curriculum was essentially an electrical engineering curriculum with a computer science emphasis. The school now graduates 15 to 20 computer science majors each year.

All computer science students are required to take a four-course sequence in software engineering. In the winter and spring quarters of the junior year, they take Software Engineering and Software System Documentation. In the fall and winter quarters of the senior year, the students take Senior Project.

The Software System Documentation course grew out of a technical writing course [Young91]. However, by placing it after software engineering in the curriculum, it rapidly became a different course from its predecessor. The maturity and experience of the students enabled the instructor to increase the difficulty of the assignments and to introduce more advanced topics to the students. During the 1990-91 academic year, the same professor taught both courses and used the opportunity to combine them to integrate software engineering concepts with the technical writing skills necessary to accomplish software engineering tasks. The combined course lasts twice as long as the Software Engineering course. Students appreciate having extra time to complete their software projects. The longer time frame also allows all software engineering assignments to be revised and polished. Students are expected to produce professional quality work at all stages of their software engineering projects. All written work is kept on-line so that documents can easily incorporate excerpts from previous work.

For more information, contact: Rose-Hulman Institute of Technology, Computer Science Department,
Terre Haute, Indiana 47803.

2.5. Drexel University

Drexel University is a private university in Philadelphia. It has a technological orientation, and all students in engineering, science, and information studies participate in a co-op program that alternates college studies and paid employment beginning in the sophomore year.

The undergraduate computer science program at Drexel included a two-quarter elective sequence in software engineering in the early 1980s. In 1983, the first course of that sequence became a required course, along with either the second course or a two-course sequence in compiler construction. The goal was to ensure that students had some team programming experience.

In the early 1990s, the curriculum was revised to include six tracks, based on the computer science categories described by Denning [Denning89]. The tracks are

- Data Structures and Algorithms
- Numerical and Scientific Computation

- Artificial Intelligence
- Operating Systems
- Programming Languages and Compilers
- Software Engineering and Methodology

Students must complete three of the tracks, including software engineering and methodology. This latter track includes four courses: Object-Oriented Programming, Software Engineering, and Software Engineering Workshop I-II. The software engineering course is taken in the junior year, and the two-course workshop sequence provides a senior year capstone experience.

Drexel has also begun offering a master's degree in software engineering, and it is expected that as that program matures, many of the graduate-level courses will be taken as advanced electives by undergraduates.

For more information, contact: Drexel University, Department of Mathematics and Computer Science, Philadelphia, Pennsylvania 19104.

2.6. State University of New York, Oswego

The State University of New York at Oswego is one of 13 colleges of arts and sciences in the New York state university system. It has more than 6000 undergraduate students. It has offered a bachelor's degree in computer science for many years.

Recently, the curriculum in computer science was modified to offer a concentration in software engineering, with the goal of giving students a strong background in software design and development in preparation for careers in the software industry [Tymann94]. A design consideration for the concentration was that it include a balance of material from software analysis, computer systems, software systems, and software process, as suggested by the SEI [Ford90].

For pragmatic reasons, the new concentration was designed to supplement the existing computer science curriculum; it required small changes to some existing courses and only a few entirely new courses. The course changes can be characterized as trying to balance the science side of computing with the engineering side, and they are being accomplished, in part, by the incorporation of laboratory work into several courses. In addition, several courses with large or group programming projects are being modified to include software engineering concerns such as documentation and modification of existing systems.

Existing software engineering courses include a junior-level introduction to software engineering and a course on systems analysis. New courses are being offered in software design and software environments. The faculty expects that as they gain more experience in teaching software engineering, they will be able to introduce additional new courses. Currently under consideration are a course in software validation and verification and a project course or practicum with real projects from industry.

For more information, contact: State University of New York at Oswego, Department of Computer Science, Oswego, New York 13126.

2.7. University of Virginia

The Department of Computer Science at the University of Virginia is in the School of Engineering and Applied Science, and the faculty feels "a strong commitment to achieving a true sense of rigorous engineering in our educational culture. We seek to educate computer scientists with a clear understanding of, an appreciation for, and skills that support the engineering and comprehension of large software systems, reengineering of existing systems, use of modern tools and environments, and application of innovative techniques such as software reuse." [Knight94].

To achieve these goals, the department has recently revised its core curriculum. The first four courses in the core are titled Introduction to Computer Science, Software Development Methods, Program and Data Representation, and Advanced Software Development. An emphasis on software engineering begins in the first course, and a philosophy of engineering is incorporated into all of the core courses [Prey94].

For more information, contact: University of Virginia, Department of Computer Science, Charlottesville, Virginia 22903.

2.8. University of West Florida

The University of West Florida, founded in 1963, is part of the state university system of Florida.

The computer science curriculum has an option called Computer Information Systems, which the department describes as "having a strong software engineering focus." The option includes a three-semester sequence Software Engineering I, Software Engineering II, and Sys'ems Project.

The first course in the sequence is an introduction to software engineering and an overview of a typical software life cycle. Early life-cycle activities are covered in somewhat more detail. The second course emphasizes software design and testing. Students in these courses often produce a software specification and design that is then implemented in the Systems Project course.

For more information, contact: University of West Florida, Department of Computer Science, Pensacola, Florida 32514.

2.9. Rochester Institute of Technology

The Rochester Institute of Technology (RIT) has offered the degree "Master of Software Development and Management" since 1987. The school's undergraduate computer science program includes a four-course sequence in software engineering beginning in the sophomore year.

In 1993, a committee of faculty from computer science and several engineering departments began developing an undergraduate program, probably to be titled "Bachelor of Science in Software Engineering." They hope to begin the program in the fall of 1995.

The RIT effort is noteworthy for several reasons.

1. It is likely to be the first such program in the United States.
2. The committee designing the program represents all the affected constituencies on campus (several departments, schools, and colleges), so that the political issues and "turf battles" can be openly addressed and resolved early in the process. The committee spent time investigating the depth and breadth of software engineering, and they were able to reach consensus that software engineering is an engineering discipline and that it is different from computer science and computer engineering.
3. The effort is likely to show the extent to which the content of an existing graduate program can be adapted in the creation of an undergraduate program.

The RIT program will also address a constraint that most schools will not have: all RIT programs are cooperative programs. The students require five years to complete the program, including a total of one year in industry. This presents an interesting problem in curriculum design. If the first co-op period is as early as the sophomore year, how should the early courses be structured to provide the students with sufficient skills to make effective use of their first industry assignment?

Engineering programs at RIT are accredited by ABET, and the school expects to seek accreditation for its software engineering program as well. Because it could be the first school to do so, it will likely help catalyze the resolution of many accreditation issues.

For more information, contact: Rochester Institute of Technology, Department of Computer Science, Rochester, New York 14623.

2.10. University of Washington

The University of Washington (UW) in Seattle has branch campuses to serve the Puget Sound region of northwest Washington. A particular goal of the branch campuses is to meet the intellectual and pragmatic needs of students whose goal is employment in the software industry.

Toward this end, a committee including faculty from the UW Department of Computer Science and Engineering, faculty from the Bothell branch campus, and industry representatives has been working to define a curriculum for the branch campuses leading to a degree in "Software Systems." The curriculum is organized as three components: preparation, core computer science, and software development processes. The preparation includes mathematics, science, writing, and programming courses.

The core computer science courses include

- Object-Oriented Programming and Abstract Data Types
- Discrete Structures and Formal Models
- Data Structures and Their Algorithms
- Computer Systems
- Programming Languages and Their Implementation
- Networks and Distributed Systems
- Databases

The software development processes courses include

- The Software Product
- Information Design and Product Presentation
- Software Production
- Requirements and Specification
- Design
- Testing, Analysis, and Verification

The curriculum is designed to teach fundamental concepts, while providing an extensive laboratory component in which the students can develop more specific, job-oriented skills.

Funding limitations have delayed the implementation of this program, although planning and development are still proceeding.

For more information, contact: University of Washington, Department of Computer Science, Seattle, Washington 98195.

2.11. Florida Institute of Technology

The Florida Institute of Technology began efforts in 1991 to offer an undergraduate software engineering degree. The faculty chose the challenging strategy of first introducing a new freshman course sequence that stressed Harlan Mills' *cleanroom* development method. All freshman computer science majors took the new sequence. The strategy called for introduction of additional software engineering courses each year until the full four-year program was in place.

By early 1993, it became evident that the strategy was not succeeding. A major factor was the difficulty in developing the necessary course materials for the new freshman sequence—materials that presented the Ada language in the context of the functional verification techniques and box-structured development strategies required by the cleanroom method. In addition, the inclusion of the new material in the freshman courses limited the students' abilities to acquire traditional programming skills. This became a significant issue for the faculty as the students entered the more traditional sophomore computer science courses.

Ultimately, the faculty decided to return to a typical freshman computer science course sequence. Some higher-level software engineering courses are being retained, including a project-oriented course.

One other lesson can be learned from Florida Tech's experience. The new freshman courses were viewed by faculty outside the department as inappropriate service courses in programming for science and engineering majors. This is likely to be a concern at many schools that consider introducing freshman-level software engineering courses. As is the case in many disciplines, the differing needs of majors and non-majors may require a department to offer different introductory courses to the two groups.

For more information, contact: Florida Institute of Technology, Department of Computer Science, Melbourne, Florida 32901.

2.12. Other Schools

In addition to the schools whose programs or plans were described in the previous sections, there are probably many others that have introduced more software engineering into their computer science programs in recent years. For example, we have heard that there are now software engineering options in the computer science programs at Washington State University and at the California Polytechnic State University.

We also note that software engineering programs have begun appearing in universities outside the United States. For example, in Australia, the Swinburne University of Technology offers a Bachelor of Science in Software Engineering and the University of Melbourne offers a Bachelor of Engineering in Software Engineering.

We solicit information from all schools for inclusion in subsequent re

3. Evolution of Undergraduate Software Engineering Programs

It may seem that the growth of distinct undergraduate software engineering degree programs is proceeding slowly. A significant question to ask at this point is whether that growth is typical—is it what we would expect to see in a new technical discipline? This chapter tries to answer that question.

Our approach is to look for trends in the growth of software engineering programs and compare them to historical trends for related disciplines. In particular, we examine the evolution of computer science and computer engineering programs. We would expect that some of the patterns of growth of those programs would suggest how software engineering programs might emerge.

Section 3.1 presents some factual data on the growth of software engineering programs in United States universities. Section 3.2 compares that growth to the growth of computer science programs in the 1960s. Section 3.3 looks at the evolution of the computer engineering program at Carnegie Mellon University, which provides an interesting example of the problem of creating separate programs in closely related disciplines.

3.1. Growth of Graduate Software Engineering Programs

We have heard suggestions that new academic disciplines tend to emerge first at the master's level. This is plausible for several reasons.

- A new discipline often evolves out of specialized areas of another discipline, and a student needs a foundation in the parent discipline before studying the new one.
- Graduate programs often have topics or seminar courses already on the books that provide the flexibility to teach new topics with a minimum of formal administrative barriers.
- Experience gained in teaching graduate courses makes it easier to organize and less risky to teach the same material at the undergraduate level.
- Developing a year-long master's curriculum is substantially easier than developing a four-year bachelor's curriculum.

To examine this hypothesis, we can first look at some statistics.

Graduate programs in software engineering first appeared in 1978 at Seattle University, and in 1979 at Texas Christian University and the Wang Institute of Graduate Studies (which closed in 1987). Over the past 15 years, nearly 25 additional programs have been developed. In addition, more than 25 other universities have a named or unnamed software engineering option or track within their graduate computer science or similar programs.

<i>MS in Software Engineering</i>	<i>MS in Software Systems Engineering</i>
Andrews University	George Mason University
Colorado Technical College	<i>MS in Software Development and Management</i>
Drexel University	Rochester Institute of Technology
Kansas State University	<i>Master of Software Design and Development</i>
Monmouth College	Texas Christian University
National Technological University	University of St. Thomas
National University	<i>MS in Systems Engineering</i>
Southern Methodist University	Boston University
University of Houston, Clear Lake	<i>MS in Software Systems Management</i>
University of Pittsburgh	Air Force Institute of Technology
University of Scranton	<i>Certificate in Software Engineering</i>
University of St. Thomas	Azusa Pacific University
<i>Master of Software Engineering (MSE)</i>	New York University
Carnegie Mellon University	Santa Clara University
Embry-Riddle Aeronautical University	Troy State University in Montgomery
Seattle University	University of Colorado at Colorado Springs
<i>Master of Engineering in Software Engineering</i>	<i>Certificate in Software Systems Engineering</i>
University of Colorado	George Mason University

Table 3.1. Graduate Programs in Software Engineering

<i>MS in Computer Science , named SE option</i>	<i>MS in Computer Info. Systems (SE emphasis)</i>
Air Force Institute of Technology	Grand Valley State University
California State University, Sacramento	<i>MS in Computer Science , unnamed SE option</i>
East Tennessee State University	Arizona State University
Florida Atlantic University	Georgia Institute of Technology
Florida Institute of Technology	Mississippi State University
George Washington University	Purdue University
Portland State University	University of Alabama in Huntsville
San Jose State University	University of Florida
University of Alaska at Fairbanks	University of Maryland
University of Iowa	University of Tennessee
University of Southern California	<i>Master of Computer Science , unnamed SE option</i>
University of West Florida	Arizona State University
<i>Master of Computer Science, Software Engineering Option</i>	
Wichita State University	

Table 3.2. Graduate Programs in Computer Science with a Software Engineering Option

These programs are not always easy to identify, primarily because not all of them have the phrase "software engineering" in their titles. For the purposes of tracking the growth of software engineering education, we have tried to look beyond the titles and include all programs that address professional education of software practitioners at the master's level (including certificate programs).

The software engineering programs in United States universities that we have been able to identify are shown in Table 3.1, grouped by program title. Descriptions of many

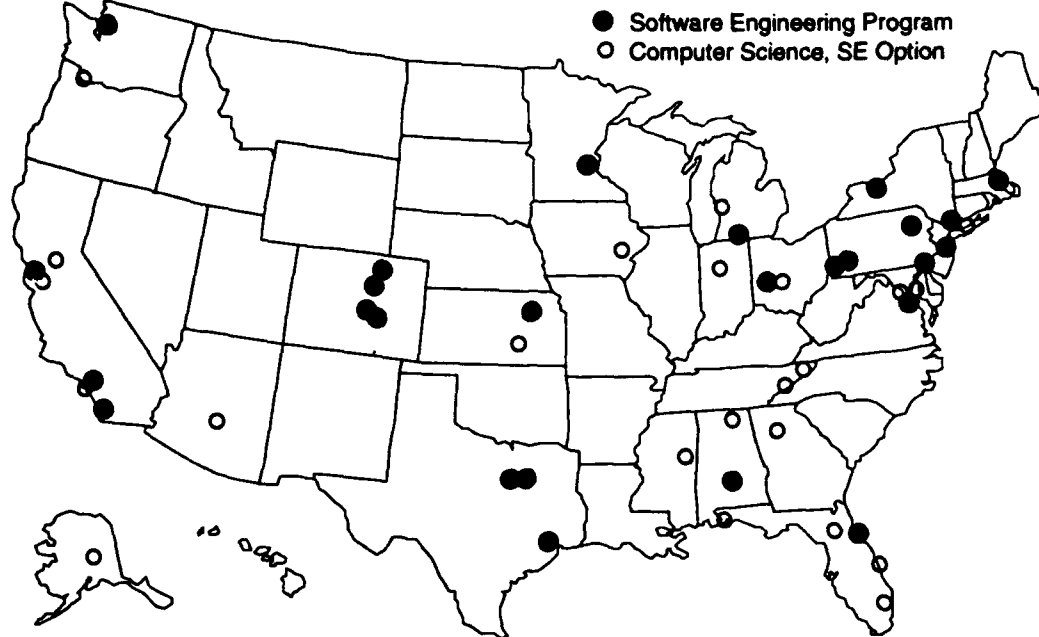


Figure 3.1. Locations of Software Engineering Graduate Programs

of these programs appear in a previous report [Ford91]. Table 3.2 lists the universities that have a software engineering option or track within a related degree program. Figure 3.1 shows the geographical distribution of all these programs.

Figure 3.2 shows graphically the pace of the growth of the graduate programs in Table 3.1; the horizontal axis is the year in which the program was officially established. We have been unable to attach precise dates to the programs listed in Table 3.2 because there was no comparable defining event for those programs. We believe, however, that the growth of those programs roughly parallels the growth curve in Figure 3.2.

3.2. Evolution of Computer Science Programs

The evolution of computer science degree programs is well documented in Seymour Pollack's excellent history of computer science education [Pollack82]. By looking at that history, we can gain insight into several questions:

- How fast did the number of computer science programs grow?
- What were the important causes of the creation of computer science programs?
- How did computer science curricula evolve?
- Did computer science programs appear first at the master's level (as discussed in the previous section)?

We may be able to identify aspects of the evolution of software engineering programs that parallel those of computer science programs. If so, they may suggest likely scenarios for the future of software engineering education.

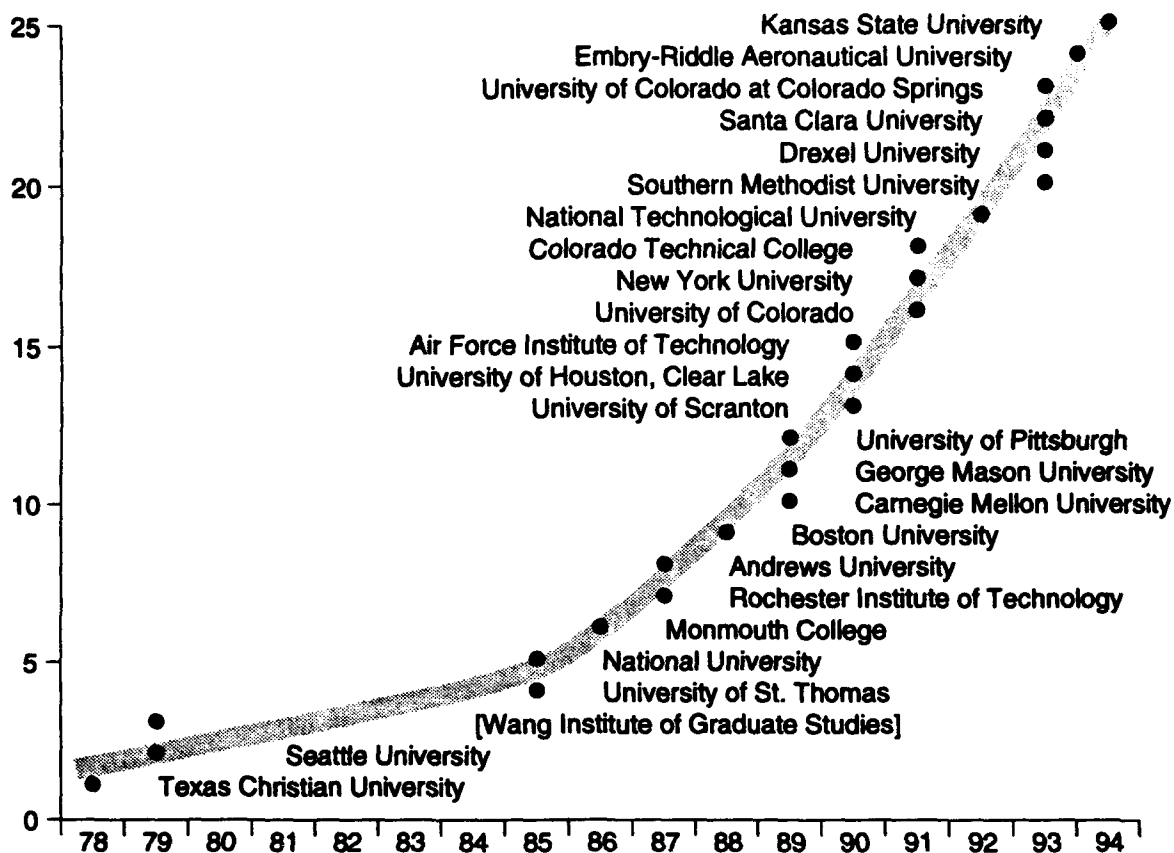


Figure 3.2 Growth of Master's Programs in Software Engineering

Growth rate. To compare the rates of growth of computer science and software engineering programs, we first consider these statistics:

- The first few PhD programs appeared about 1961; these were interdisciplinary programs in existing departments rather than separate degree programs.
- By 1964 there were about a dozen computer science bachelor's degree programs in US universities.
- Between 1964 and 1968, the number of bachelor's programs grew to nearly 100; master's programs experienced a similar increase. The number of PhD programs grew from about 10 to about 40.

Compared to computer science, the growth of separate software engineering programs has been much slower. Although there have been calls for such programs as early as 1969 [Kuo69], no undergraduate programs actually named *software engineering* have been created (see the previous chapter for some descriptions of programs that come close). We do believe, however, that the majority of computer science programs now have at least one course in software engineering.

In the previous section, we noted that approximately 25 master's or certificate programs in software engineering have been created between 1978 and 1994—a 16-year span.

Figure 3.2 shows the growth of those programs over that period. This growth rate is also much slower than that of computer science.

We do not know of doctoral programs in software engineering that are so-named and separate from computer science programs. Many universities report that increasing numbers of students in computer science doctoral programs are writing dissertations on software engineering topics. Prominent among these in the last several years have been Arizona State University, Purdue University, the University of Florida, the University of Maryland, and the University of California at Irvine.

Causes of program development. One catalyst of the rapid growth of undergraduate computer science programs between 1964 and 1968 was the publication of the first curriculum recommendations [ACM65] from the Association for Computer Machinery (ACM). We note that neither the ACM nor the IEEE Computer Society (the other main professional society) has yet published a model curriculum for an undergraduate software engineering degree (however, see the next chapter). We believe such a model curriculum would be an important factor in the growth of undergraduate software engineering programs, just as was the case for computer science.

The Software Engineering Institute published its first master's curriculum recommendations in 1987. Although the rate of growth of master's programs increased (see Figure 3.2), the SEI curriculum was not the only factor in that growth. The increasing needs of industry for educational opportunities for software engineers was a major reason for the development of new programs. On the other hand, the majority of programs started since 1987 acknowledge having been influenced by the SEI curriculum. This reinforces our belief that a model curriculum is an important catalyst for creation of new programs.

Curriculum evolution. One of the principal reasons for advocating the development of software engineering programs is the perception that computer science curricula have evolved to a state that does not adequately prepare students for professional careers building software-intensive systems. How did computer science curricula get to that state?

Model curricula greatly influence the philosophy and direction of degree programs. Pollack notes that the most significant event in the early history of computer science education, the publication of ACM *Curriculum '68* [ACM68], led the education community further toward science (and research) and away from engineering (and professional practice):

Interestingly, Curriculum '68's influence also had a dichotomizing aspect: Its basically mathematical orientation sharpened its contrast with more pragmatic alternatives. Most computer science educators agreed that the proposed core courses included issues crucial to computer science. However, the curriculum brought to the surface a strong division over the way in which these issues should be viewed. In defining the contents of the courses, Curriculum '68 established clearly its alignment with more traditional mathematical studies, giving primary emphasis to a search for beauty and elegance. Pedagogically, this implied a set of academic objectives concerned chiefly with preparation for

graduate study leading to a career in research. Consequently, those colleges and universities holding with the perception of computer science saw Curriculum '68 as a reinforcement and endorsement of their orientation and sought to implement it commensurate with their resources.

On the other hand, many educators felt the curriculum to be at odds with their perception of reality. They argued that the uses of computer science and the observed roles of computer scientists militate for an education approach much closer to that used in professional disciplines. ... In this light, computer science education should have a strong professional flavor (it was argued), with design principles, general approaches to problem solving, and experiments with current methodologies receiving considerable attention. This would be consistent with the expectation of professional employment starting at the baccalaureate level.

Two arguments heard today for not having software engineering programs are these:

- Software engineering is not sufficiently mature to warrant a separate college curriculum.
- Software engineering is a specialty area that should only be taught at the graduate level after a student has an undergraduate background in computer science.

Pollack notes that similar arguments were made in the 1960s regarding computer science:

At a more fundamental level, many universities, while convinced of computer science's separate identity, felt that an independent program was premature. For them, computer science was a graduate specialty to be preceded by undergraduate concentration in some established area (not necessarily mathematics or science).

The fact that more than 900 United States colleges and universities now offer some kind of computing-related undergraduate degree suggests that these arguments against separate computer science programs were not valid. Neither do we find them to be persuasive when applied to software engineering.

It seems more plausible that the slower growth of software engineering education can be attributed to the availability of computer science education, which for two decades has produced graduates with computer programming knowledge and thus has reduced the urgency of the need for software engineering education. Only recently, as society has become increasingly dependent on complex, software-intensive systems, has the demand for software engineers grown significantly, resulting in greater appreciation for the differences between the two disciplines.

These differences can be traced to seeds planted in the early 1960s. Pollack notes

The rapid growth of computer science education stimulated increased interest in theoretical areas (such as automata theory and formal languages) whose pursuit predated computers. Now, these areas were seen potentially to impinge on questions raised by the design and use of computer systems. Consequently, there appeared to be a prospect of concurrent and mutually nourishing development in computer science theory and practice. Curiously, this did not happen. The newly intensified effort generally maintained its own paths, interacting very

little with the application-motivated problems that were helping to spur head-long advances in hardware and software technology.

In considering this rapid growth of the theoretical currents of computing, we are reminded of the seemingly prophetic remarks of computing pioneer John von Neumann [vonNeumann47]:

As a mathematical discipline travels far from its empirical source, or still more, if it is a second- and third-generation only indirectly inspired from ideas coming from "reality," it is beset with very grave dangers. It becomes more and more purely aestheticizing, more and more purely *l'art pour l'art*. This need not be bad, if the field is surrounded by correlated subjects, which still have closer empirical connections, or if the discipline is under the influence of men with an exceptionally well-developed taste.

But there is a grave danger that the subject will develop along the line of least resistance, that the stream, so far from its source, will separate into a multitude of insignificant branches, and that the discipline will become a disorganized mass of details and complexities. ... [W]henver this stage is reached, the only remedy seems to me to be the rejuvenating return to the source: the reinjection of more or less directly empirical ideas. I am convinced that this is a necessary condition to conserve the freshness and the vitality of the subject, and that this will remain so in the future.

A typical undergraduate computer science curriculum of the last ten to twenty years exhibits some of the danger suggested by von Neumann; that is, the curriculum includes several independent, one-semester courses that cover individual branches of the discipline. Furthermore, the most common courses that *do* reach back toward the empirical source (building useful artifacts) are about operating systems and compilers—artifacts from the computer science domain itself.

First programs at the master's level. In the previous section we noted the hypothesis that degree programs in a new discipline first emerge at the master's level. Although this seems to be happening in software engineering, the statistics cited above on the growth of degree programs during the 1960s tend to refute that hypothesis for computer science.

Possible futures. Pollack (writing in 1982) concludes

This ongoing turmoil, fueled by a diversity of viewpoints, will continue to enrich the discipline and could well lead to convergence on not one but several viable identities.

We believe that the education community is on the verge of realizing Pollack's prediction. The mathematical, abstract, science-like, and research-oriented aspects of computing will continue to be addressed by programs in computer science. The application of computing knowledge in the building of useful, software-intensive products will be addressed by programs in software engineering. In addition, computation (for purposes of simulation and visualization) has emerged as a third paradigm of science (along with theory and experimentation), leading to the growth of computational science programs.

We can only imagine how computing will ultimately affect education in the professional disciplines and in the humanities, social sciences, and fine arts.

3.3. The Computer Engineering Program at Carnegie Mellon University

Many educators argue that computer science and software engineering are too closely related to support separate undergraduate degree programs. A similar argument can be made for electrical engineering and computer engineering. We know that some schools have combined electrical and computer engineering programs and some schools have separate programs. Thus it may be useful to look at the evolution of a computer engineering program in order to support or refute that argument. The Department of Electrical and Computer Engineering (ECE) at Carnegie Mellon University (CMU) provides an instructive case study.

With the growth of computer technology in the 1960s and 1970s, and especially with the advent of VLSI (very large scale integration) technology and microcomputers, electrical engineering departments found themselves challenged to include new topics and new courses in an already broad curriculum. CMU ECE associate department head James Hoburg reflected on their solution to this problem [Hoburg90]:

Breadth of understanding across an engineering discipline is not the ultimate good thing for all students. Some may actually be better served by sacrificing breadth to depth of understanding in a subset of the areas which constitute the discipline.

Several years ago, we peeled off Computer Engineering from Electrical Engineering as a separate discipline with a separate degree program. Suddenly, much of what seemed so essential to all electrical engineers was no longer essential when the Computer Engineering option became the Computer Engineering degree. By inventing a separate discipline, we no longer felt obliged to require breadth across all electrical technologies. This also helped to alleviate the pressure we felt to incorporate rapidly expanding digital technologies into the Electrical Engineering curriculum. Since those technologies were the heart of Computer Engineering, now a separate discipline, it was no longer essential for all electrical engineers to know all about them.

Figures 3.3a and 3.3b show the separate electrical engineering and computer engineering curricula (as presented in the 1988-1990 CMU catalog). The numbers indicate units of credit (3 units are essentially equivalent to 1 semester hour; both curricula require 388 units, or approximately 129 semester hours). Note that the freshman and sophomore years are virtually identical in the two programs, while the junior and senior years differ.

Both curricula exhibit an almost stereotypical characteristic of engineering curricula: a very high percentage of specific required courses in a lockstep sequence. More specifically, the electrical engineering curriculum has 51% of the units in specific required

Curricula at Carnegie Mellon University - 1988	
Electrical Engineering	Computer Engineering
<i>Fall, Freshman Year</i>	<i>Fall, Freshman Year</i>
10 Calculus	10 Calculus
10 Physics I: Mechanics	10 Physics I: Mechanics
10 Modern Chemistry I	10 Modern Chemistry I
3 Computing Skills Workshop	3 Computing Skills Workshop
9 Designated Writing Course	9 Designated Writing Course
9 Freshman Elective	3 Freshman Elective
<i>Spring, Freshman Year</i>	<i>Spring, Freshman Year</i>
10 Calculus with Linear Algebra	10 Calculus with Linear Algebra
10 Physics II: Heat, Wave Motion and Optics	10 Physics II: Heat, Wave Motion and Optics
10 Introduction to Computing for Engineering & Science	10 Introduction to Computing for Engineering & Science
3 Freshman Chemistry Lab	3 Freshman Chemistry Lab
9 Humanities/Social Science	9 Humanities/Social Science
9 Freshman Elective	9 Freshman Elective
<i>Fall, Sophomore Year</i>	<i>Fall, Sophomore Year</i>
9 Differential Equations	9 Differential Equations
10 Physics III: Electricity and Magnetism	10 Physics III: Electricity and Magnetism
12 Introduction to Digital Systems	12 Introduction to Digital Systems
9/11 Engineering Science Elective	9 Introduction to Modern Mathematics
9 Humanities/Social Science	9 Humanities/Social Science
<i>Spring, Sophomore Year</i>	<i>Spring, Sophomore Year</i>
9 Calculus in Three Dimensions	9 Calculus in Three Dimensions
12 Linear Circuits	12 Linear Circuits
12 Intro to Electronic Devices and Circuits	12 Intro to Electronic Devices and Circuits
9/11 Engineering Science Elective	9 Fund Structures of Computer Science I
9 Humanities/Social Science	9 Humanities/Social Science

Figure 3.3a. CMU ECE Curriculum: Freshman and Sophomore Years

courses; the computer engineering curriculum has 55%. Both curricula require 75% of the units in technical courses.

A situation similar to that faced by the CMU ECE faculty in the mid-1980s now faces computer science faculty. They are challenged to include in their computer science programs the large and growing body of knowledge of software engineering (and perhaps some of the other growing areas such as computational science). The most common path for curriculum evolution (because it is the easiest to implement) is to incorporate new topics into the curriculum by adding new elective courses at the senior level. This approach tends to sacrifice depth for breadth, leading back to Hoburg's argument above.

The CMU ECE experience suggests that separate curricula in computer science and software engineering (and perhaps also computational science) would allow sufficient depth in each area to satisfy the various professional education goals (graduate study, software development career, etc.) of the majority of students.

Curricula at Carnegie Mellon University - 1988	
Electrical Engineering	Computer Engineering
<i>Fall, Junior Year</i>	<i>Fall, Junior Year</i>
9 Engineering Electromagnetics I	12 Introduction to Computer Architecture
12 Analysis and Design of Digital Integrated Circuits	12 Analysis and Design of Digital Integrated Circuits
9 Signals and Systems I	9 Fund Structures of Computer Science II
9/12 Technical Elective	9/11 Engineering Science Elective
9 Humanities/Social Science/Arts	9 Humanities/Social Science/Arts
<i>Spring, Junior Year</i>	<i>Spring, Junior Year</i>
9 Engineering Electromagnetics II	12 Concurrency and Real Time Systems
9 Signals and Systems II	9/11 Engineering Science Elective
12 Analysis and Design of Analog Circuits	9 Technical Elective
9/12 Technical Elective	9/12 Technical Elective
9 Humanities/Social Science/Arts	9 Humanities/Social Science/Arts
<i>Fall, Senior Year</i>	<i>Fall, Senior Year</i>
0 ECE Senior Seminar	0 ECE Senior Seminar
9 Probability for Electrical Engineers	9 Probability for Electrical Engineers
12 Senior Design Elective	12 Senior Design Elective
12 Technical Elective	12 Logic and Processor Design
9 Humanities/Social Science/Arts	9 Humanities/Social Science/Arts
<i>Spring, Senior Year</i>	<i>Spring, Senior Year</i>
9/12 Senior EE Elective	9/12 Technical Elective
12 Technical Elective	9/12 Technical Elective
9/12 Technical Elective	9/12 Technical Elective
9 Free Elective	9 Free Elective
9 Humanities/Social Science/Arts	9 Humanities/Social Science/Arts

Figure 3.3b. CMU ECE Curriculum: Junior and Senior Years

However, this is not the end of the CMU ECE story. In the last several years, a lot of thought has been devoted to improving professional education in general, and engineering education in particular. Many engineering departments are solving the breadth vs. depth problem by increasing the flexibility of their programs. The cost is *an increased commitment by the faculty to do meaningful advising* of students, but the benefit is an opportunity for each student to choose a program of study that matches his or her professional goals.

It was with this approach in mind that the CMU ECE faculty undertook a complete curriculum review and redesign, resulting in a single degree program in electrical and computer engineering. A comparison of the old (separate) and new curricula yields some interesting insights.

Figure 3.4 shows the new combined curriculum (as presented in the 1992-1994 CMU catalog). It requires a total of 360 units (120 credit hours), of which only 33% is in specific courses. The core of this curriculum consists of only three engineering courses: Introduction to Electrical and Computer Engineering, Fundamentals of Electrical Engineering, and Fundamentals of Computer Engineering. The latter two of these courses have mathematics corequisites: Linear Algebra and Introduction to Modern

Curriculum at Carnegie Mellon University - 1994	
Electrical and Computer Engineering	
<i>Fall, Freshman Year</i>	<i>Fall, Junior Year</i>
12 Intro to Electrical and Computer Engineering	12 ECE Breadth Course 1
10 Intro to Programming and Computer Science	12 ECE Breadth Course 1
10 Calculus	12 Free Elective
9 Designated Writing Course	9 Math Elective
3 Computer Skills Workshop	9 Humanities/Social Science
<i>Spring, Freshman Year</i>	<i>Spring, Junior Year</i>
12 Introduction to Engineering Elective	12 ECE Depth Course
12 Physics for Engineering Students I	12 ECE Coverage Course 1
10 Calculus with Linear Algebra	12 Free Elective
9 Humanities/Social Science	9 Humanities/Social Science
<i>Fall, Sophomore Year</i>	<i>Fall, Senior Year</i>
12 ECE Core Course	12 ECE Coverage Course 2
9 Restricted Math Elective	12 Free Elective
12 Physics for Engineering Students II	12 Free Elective
9 Humanities/Social Science	9 Humanities/Social Science
<i>Spring, Sophomore Year</i>	<i>Spring, Senior Year</i>
12 ECE Core Course	12 Free Elective
9 Restricted Math Elective	12 Free Elective
12 ECE Breadth Course 1	12 Free Elective
9 Humanities/Social Science	9 Humanities/Social Science

Figure 3.4 CMU Combined Electrical and Computer Engineering Curriculum

Mathematics, respectively. Students must satisfy a breadth requirement by taking one course in at least three of the principal subject areas: physics, signals and systems, circuits, computer hardware, and computer software. A depth requirement requires a second course in one of the breadth areas. Two additional ECE courses are required, plus a capstone design elective.

This approach to an engineering curriculum may also provide a model for departments wanting to offer both computer science and software engineering curricula. The lesson to be learned is very important, however. To achieve the capability to serve both electrical engineering and computer engineering students, the CMU ECE faculty *redesigned the introductory and core courses specifically for the combined curriculum*. They did not simply add a lot of junior- and senior-level specialty courses on top of a traditional electrical engineering core or a traditional computer engineering core.

Hoburg expresses some of the philosophy behind the redesign of the core [Hoburg90]:

Don't assume that all engineering must be postponed until a foundation of science and mathematics is laid. Teach more engineering and less basic science and mathematics early in a curriculum. Provide motivation early by exposing students to the applications which we too often implicitly assume they already know about but which, in fact, are completely foreign to many. ...

Provide rich and deep courses in fundamentals, basic science and mathematics, and real understanding at the one-to-two-years-before-graduate-school level for students to take late, not early, in the curriculum. Don't require all students to take all such courses. Rather, help students, through high quality advising, to pursue individual mixtures of breadth and depth based upon their individual abilities and interests.

We expect that many schools will develop undergraduate curricula that attempt to satisfy the needs of both computer science and software engineering students. As was the case for the CMU ECE curriculum, the biggest challenges are in the design of the introductory courses. In most current curricula, the idea is deeply ingrained that the introductory courses must teach programming as an ad hoc, individual process. Changing these courses to provide a proper basis for the engineering of software will be technically difficult and will present political problems in schools where the introductory courses are also service courses for other disciplines.

Recent statistics suggest that less than 10% of computer science students go immediately to graduate school; the vast majority take jobs involving the development of software. For this reason, we would hope that any combined computer science and software engineering curricula of the future would follow Hoburg's advice above: provide the engineering early in the curriculum. Elective courses later in the curriculum can provide the depth in the science of computing that is needed by students heading to graduate school and research careers.

3.4. Conclusions

Our brief look at the evolution of computer science and computer engineering programs leads us to four conclusions about the future evolution of undergraduate software engineering education.

1. We do not expect software engineering programs to emerge as rapidly as computer science programs did. Much of the software industry still relies on relatively undisciplined development processes and is satisfied with the level of programming skills in graduates of existing computer science programs. As the industry matures, there will be a greater demand for software engineers (rather than programmers), but that is a slow process.
2. Although separate software engineering programs in U. S. universities are appearing first at the graduate level, this is not a requirement. It is possible for a university to develop an undergraduate program without first creating a graduate program.
3. The publication by the professional societies of a model curriculum for a bachelor of science in software engineering degree would probably accelerate the growth of such programs significantly. The effects of such a model curriculum would include establishing the credibility of such programs, encouraging authors and publishers to create the needed new textbooks, and providing a basis for future accreditation guidelines for such programs.

4. For the immediate future, the most likely evolutionary path will be the creation of a software engineering track within a computer science program. We hope that schools following this path will learn from examples such as that described in the previous section (the CMU ECE curriculum) and develop introductory courses that serve *both* software engineering and computer science tracks equally well.

4. Recent Actions by the Professional Societies

Although there is no professional society in the United States specifically for software engineers, both the Computer Society of the Institution of Electrical and Electronics Engineers (IEEE-CS) and the Association for Computer Machinery (ACM) have a large percentage of their memberships who consider themselves software engineers. Both societies address the needs of software engineers through their publications and conferences.

In 1993, efforts began in both societies to address issues related to the establishment of the profession of software engineering. Professional education for software engineers is one of those issues. The next two sections summarize the societies' efforts.

4.1. IEEE Computer Society

At its May 1993 meeting, held in conjunction with the International Conference on Software Engineering (ICSE), the IEEE Computer Society Board of Governors considered a motion from Fletcher Buckley "to initiate the actions to establish software engineering as a profession." The motion identified four specific actions to be included in that work:

1. Determining appropriate definitions and establishing those definitions as IEEE approved standards.
2. Determining the body of knowledge appropriate for an undergraduate program in software engineering and establishing ABET accreditation guidelines for such programs.
3. Defining a code of ethics for software engineers.
4. Encouraging the states to adopt licensing procedures for software engineers.

Buckley's proposal was widely circulated for three months before the meeting, so many people came to ICSE prepared to discuss it. Several informal discussion sessions preceded the meeting of the Board of Governors.

Ultimately, the Board of Governors adopted a somewhat different motion, calling for a more deliberate consideration of the issues by an ad hoc committee of "well-respected individuals ... with a balance of industry, research, and academic backgrounds." The committee was to be charged with considering and documenting both the factors involved in and the value of

1. establishing software engineering as an approved academic program including the associated accreditation issues;
2. establishing a separate set of software engineering ethics;

3. establishing software engineering as a certified or registered field.

The committee was to make its initial report to the Board of Governors at their November 1993 meeting.

Computer Society president James Aylor selected a committee consisting of Mario Barbacci (Software Engineering Institute), Fletcher Buckley (Martin Marietta), Larry Druffel (Software Engineering Institute), Winston Royce (TRW), and Norman Schneidewind (U. S. Naval Postgraduate School). Stuart Zweben (Ohio State University), the current ACM vice-president, represented the ACM (see the next section).

The committee quickly recognized that it would be unable to make comprehensive recommendations or definitive proposals by the November reporting date. They agreed instead to define a process and agenda for various working groups and task forces that would consider individual issues.

Four recommendations were made by the committee at the November Board of Governors meeting [Barbacci94].

1. The Computer Society, through its Standards Activities Board and appropriate standards subcommittees, should adopt standard definitions necessary for defining a software engineering profession.
2. The Computer Society should identify the body of knowledge and the recommended practices for professional software engineers. This effort should be carried out by a task force of industry experts.
3. The Computer Society should study and customize, if necessary, existing codes of ethics, in order to develop a code of ethics for the software engineering profession. This task should be charged to the Committee on Public Policy.
4. Curricula should be defined for undergraduate, graduate (master's level), and continuing education programs in software engineering. This task should be charged to a task force drawn from the education boards or groups of the SEI, ACM, IEEE Computer Society, and other relevant societies.

The fourth recommendation included this elaboration: "There is a debate as to whether Software Engineering is a part of Computer Science or vice versa. We should not be distracted by this debate from the goal of meeting the needs of industry. The education needed by competent software engineers could be acquired in different ways. ... The objective is to seek agreement on the curricula that should be taught and not necessarily on which departments teach it."

The committee stated its belief that there are implied dependencies among the four recommended actions that would require them to be implemented in the order stated. The committee expects its next report, scheduled for the spring of 1994, to include more details of the recommendations and plans and schedules for implementing them.

Finally, the committee stated, "The process should be open to the full community. This requires early dissemination of proposals and engaging the appropriate communities."

4.2. ACM

The ACM also began efforts in 1993 to address issues related to establishing software engineering as a profession. At its August 1993 meeting, the ACM Council endorsed the establishment of a "Commission on Software Engineering" that would make recommendations regarding terminology, standards of good practice, and education and training for professional software engineers.

The ACM Council directed the commission to carry out its work jointly with the IEEE Computer Society, if possible (see the previous section). The commission was to be drawn from leading people in the software engineering community, with representation from industry, government, and academia. It would gather information from other knowledgeable groups and related published material, and it would cooperate with other organizations interested in the subject.

In particular, the commission was charged with addressing several questions, including

1. What activities are usually considered part of software engineering?
2. What is the state of software engineering as a profession?
3. What standard practices currently exist for software engineering?
4. What are the implications of the above for education? Curriculum? Accreditation?
5. What are the implications of the above for certification? Licensing?

The commission was asked to complete its work by April 30, 1994.

There are obvious similarities in the questions being addressed by the two societies. Rather than proceeding independently, the ACM and the Computer Society agreed in March 1994 to cooperate in this work.

4.3. Task Force Activities

Following the November 1993 meeting of the IEEE Computer Society Board of Governors, efforts began to create task forces to carry out the recommendations of the steering committee. A group led by Dr. Patricia Douglas (IBM Skill Dynamics) has started defining procedures to identify the body of knowledge and skills of software engineering (recommendation 2). Several computing professionals knowledgeable in the area of ethics have been asked to participate in a second task force (recommendation 3). The chairs of the IEEE Computer Society Educational Activities Board and the ACM Education Board have agreed to lead the creation of appropriate curriculum task forces (recommendation 4).

References

- ACM65 ACM Curriculum Committee on Computer Science. "An Undergraduate Program in Computer Science—Preliminary Recommendations." *Comm. ACM* 8, 9 (Sept. 1965): 543-552.
- ACM68 ACM Curriculum Committee on Computer Science. "Curriculum 68: Recommendations for the Undergraduate Program in Computer Science." *Comm. ACM* 11, 3 (Mar. 1968): 151-197.
- Ardis89 M. Ardis and G. Ford. *1989 SEI Report on Graduate Software Engineering Education* (Technical Report CMU/SEI-89-TR-21, ADA219018). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1989.
- Barbacci94 M. Barbacci. "Panel Presents First Four Recommendations Aimed at Establishing Software Engineering as a Profession." *Computer* 27, 2 (Feb. 1994): 80-81.
- Denning89 P. J. Denning; D. E. Comer; D. Gries; M. C. Mulder; A. Tucker; A. J. Turner; and P. R. Young. "Computing as a Discipline." *Comm. ACM* 32, 1 (Jan. 1989): 9-23.
- Ford90 G. Ford. *1990 SEI Report on Undergraduate Software Engineering Education* (Technical Report CMU/SEI-90-TR-3, ADA223881). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1990.
- Ford91 G. Ford. *1991 SEI Report on Graduate Software Engineering Education* (Technical Report CMU/SEI-91-TR-2, ADA236340). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1991.
- Hoburg90 J. F. Hoburg. "Thoughts on Engineering Education by a Newly Converted "Radical"." *Focus* 20, 1 (Sept. 1990): 2. Focus is a publication of the faculty and staff of Carnegie Mellon University.
- Jovanovic92 V. Jovanovic; E. Likine; and D. Shoemaker. *Model Undergraduate and Graduate Programs in Software Management* (Working Paper 92-7). Detroit, Mich.: College of Business and Administration, University of Detroit Mercy, 1992.
- Knight94 J. C. Knight; J. C. Prey; and W. A. Wulf. "Undergraduate Computer Science Education: A New Curriculum Philosophy & Overview," 155-159. *Papers of the 25th SIGCSE Technical Symposium on Computer Science Education*, D. Joyce, ed. New York: ACM Press, Mar. 1994.

- Kuo69 F. F. Kuo. "Let's Make our Best People into Software Engineers and not Computer Scientists." *Computer Decisions* 1, 2 (Nov. 1969): 94.
- Pollack82 S. V. Pollack. "The Development of Computer Science," 1-51. *Studies in Computer Science*, S. V. Pollack, ed. Washington, D. C.: The Mathematical Association of America, 1982. Vol. 22 of *Studies in Mathematics*.
- Prey94 J. C. Prey; J. P. Cohoon; and G. Fife. "Software Engineering Beginning in the First Computer Science Course," 359-374. *Software Engineering Education; 7th SEI CSEE Conference*, J. L. Díaz-Herrera, ed. New York: Springer-Verlag, Jan. 1994.
- Tymann94 P. T. Tymann; D. Lea; and R. K. Raj. "Developing An Undergraduate Software Engineering Program in a Liberal Arts College," 276-280. *Papers of the 25th SIGCSE Technical Symposium on Computer Science Education*, D. Joyce, ed. New York: ACM Press, Mar. 1994.
- vonNeumann47 J. von Neumann. "The Mathematician," 180-196. *The Works of the Mind*, R. B. Heywood, ed. Chicago: University of Chicago Press, 1947.
- Young91 F. Young. "Software Engineering and Software Documentation: a Unified Long Course," 593-595. *Proc. 21st Frontiers in Education Conf.* Sept. 1991.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None																
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited																
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A																		
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-94-TR-11		5. MONITORING ORGANIZATION REPORT NUMBER(S) ESC-TR-94-011																
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute	6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office																
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213		7b. ADDRESS (city, state, and zip code) HQ ESC/ENS 5 Eglin Street Hanscom AFB, MA 01731-2116																
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office	8b. OFFICE SYMBOL (if applicable) ESC/ENS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003																
8c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213		10. SOURCE OF FUNDING NOS. <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <tr> <td style="width: 25%;">PROGRAM ELEMENT NO</td> <td style="width: 25%;">PROJECT NO.</td> <td style="width: 25%;">TASK NO</td> <td style="width: 25%;">WORK UNIT NO.</td> </tr> <tr> <td>63756E</td> <td>N/A</td> <td>N/A</td> <td>N/A</td> </tr> </table>		PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.	63756E	N/A	N/A	N/A							
PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.															
63756E	N/A	N/A	N/A															
11. TITLE (Include Security Classification) A Progress Report on Undergraduate Software Engineering Education																		
12. PERSONAL AUTHOR(S) Gary Ford																		
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM TO	14. DATE OF REPORT (year, month, day) May 1994	15. PAGE COUNT 30 pp.															
16. SUPPLEMENTARY NOTATION																		
17. COSATI CODES <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr> <th style="width: 33%;">FIELD</th> <th style="width: 33%;">GROUP</th> <th style="width: 33%;">SUB. GR.</th> </tr> </thead> <tbody> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </tbody> </table>		FIELD	GROUP	SUB. GR.													18. SUBJECT TERMS (continue on reverse of necessary and identify by block number) curriculum software engineering curriculum education software engineering education software engineering	
FIELD	GROUP	SUB. GR.																
19. ABSTRACT (continue on reverse if necessary and identify by block number) The current status of undergraduate software engineering education in United States universities is summarized, including descriptions of programs at eleven schools. Possible scenarios for the further evolution of undergraduate software engineering programs are described, based on observations of the evolution of computer science and computer engineering programs. Recent and ongoing activities of the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM) regarding the establishment of the profession of software engineering are described, including the expected implications for undergraduate software engineering education. <p style="text-align: right; margin-top: 10px;">(please turn over)</p>																		
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS <input checked="" type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution																
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF		22b. TELEPHONE NUMBER (include area code) (412) 268-7631	22c. OFFICE SYMBOL ESC/ENS (SEI)															

ABSTRACT — continued from page one, block 19