EDUCATIONAL ISSUES IN SOFTWARE ENGINEERING

Richard E. Fairley
Associate Professor
Computer Sceince Department
Colorado State University
Fort Collins, Colorado 80523

Introduction

The term "software engineering" came into common usage as a result of the NATO Workshops on Software Engineering in 1968 and 1969 (1). At that time the term was intentionally chosen as a provocation rather than as an indication of actual practice. During the intervening decade software engineering has evolved from a wish into a major subdiscipline of computer science and engineering. Although much reamins to be done, a body of knowledge and a set of methodological guidelines are emerging which embody the application of traditional engineering values to the production and maintenance of software systems.

The statement of scope of the IEEE Transactions on Software Engineering provides a de facto definition of software engineering:

> "The scope of this journal will cover all areas which form the middle ground that lies between the intiial step of basic research and the end use of computer software. More specifically, it includes the following areas: requirement analysis and specification, programming methodology, software testing and validation, performance and design evaluations, software project management, and programming tools and standards."

The demand for individuals trained in the skills of software engineering becomes more acute as computing systems become more numerous, more complex, and more ingrained into modern society. In recognition of this demand, the IEEE Computer Society is sponsoring the development of model curricula in software engineering. The subcommittee charged with developing the curricula is preparing recommendations for both undergraduate and graduate programs in software engineering. The curricula cover the core areas of software engineering, namely:

Computer Science and Engineering
Software Methodology
Management and Communication Skills

Computer Science and Engineering comprises the fundamental concepts of hardware and software, as well as the analytical problem solving skills acquired in the study of mathematics and theory of computation. Software Methodology is the technical essence of software engineering. It includes software life cycle concepts, the associated methodologies, the software development tools and techniques. Because software engineering is a labor intensive activity, Management and Communication Skills play a central (indeed, crucial) role. In this area, we group management science, porject management techniques, technical communication, and the legal aspects of software engineering.

This paper briefly outlines the curricular efforts of the Model Curricula Subcommittee; however, the major thrust of the paper is a discussion of the issues surrounding the development of educational programs in software engineering.

The Undergraduate Curriculum

In order to satisfy diverse needs, the following approach is advocated:

1. The undergraduate curriculum will consist of a core of material that can be implemented in diverse educational environments, plus additional material which can be implemented at the option of local institutions that desire a full scale undergraduate curriculum.
2. The core will provide sufficient preparation for the Master's program in software engineering, but will not, in itself, provide adequate training for a professional software engineer.
3. The undergraduate core will be an implementation based on the existing CSE Model Curriculum (2).

In 1977, the Model Curriculum Subcommittee of the Education Committee of the Computer Society published a curriculum in computer science and engineering which was intended to bridge the gap between hardware and software. The present curricular effort in software engineering is an outgrowth of that subcommittee's work, and the undergraduate core will be based on their report.

Figure 1 illustrates the structure of the proposed undergraduate core. The content of the courses is specified by referencing them to the corresponding courses in the Model Curriculum

Subcommittee Report.

Core Course	Reference Course(s)
DL - Digital Logic	DL-1, DL-2
DLab - Digital Lab	L-1
CO - Computer Organization	CO-1, CO-2, CO-3
MP - Microprocessors	DL-3
MPLab	L-3
ICP - Introduction to Computer Programming	SE-1
DSDA - Data Structures and Design of Algorithms	SE-2, SE-3, TC-2
OSCA - Operating Systems and Computer Architecture	SE-6, SE-7
DBS - DataBase Systems	SE-4
SLC - Survey of Language Concepts	SE-5
LI - Language Implementation	SE-8
SDTO - Software Development Tools	SE-8
SDTE - Software Development Techniques	
DS - Discrete Structures	TC-1
FLA - Formal Languages and Automata	TC-3

In addition, it is expected that a student would complete a mathematics sequence consisting of differential and integral calculus, linear algegra, and probability and statistics. The additional material needed to implement a full undergraduate program in software engineering is being developed and will be presented in the committee report. Note, for example, that there are no management or communication skills courses prescribed in Figure 1.

The Graduate Curriculum

Specifying a professional Master's degree program in software engineering is the primary emphasis of our work. A graduate of the Master's program should possess the following skills:

1. Broad based familiarity with computer science and engineering concepts such as digital logic, computer organization, machine architecture, operating systems, data structures, database systems, programming languages, language implementation, discrete mathematics, formal languages and automata, telecommunications, distributed processing, man-machine interaction, and performance measurement and evaluation -- in short, the undergraduate core plus additional course work at the graduate level.
2. The ability to use state-of-the-art tools and techniques for analysis, design, implementation, validation, maintenance, and documentation of software systems.
3. The ability to manage small programming teams (3 to 7 persons) and to provide technical leadership for programming projects of long duration.
4. The ability to communicate, in both oral and written form, with users, managers, programmers, and persons in other technical disciplines.

In order to achieve these objectives, we have identified the graduate level topics in each of the three core areas:

Computer Science and Engineering

- System Design
- DataBase Systems
- Distributed Processing
- Man-Machine Interaction
- Performance Measurement and Evaluation

Software Methodology

- Problem Definition
- Requirements Analysis
- Design Techniques
- Programming Methodology
- Validation and Verification
- Maintenance
- Tools
- Lab Sequence

Management and Communication Skills

- Project Management Techniques
- Management Science
- Technical Communication
- Legal Aspects of Software Engineering

It should be emphasized that these topics are not course titles, but rather modules that will be elaborated and may become course or modules within courses.

Ideally, a student entering the Master's program would have completed the undergraduate core courses, plus two to three years work experience. In reality, most students will have a more traditional computer science degree and no significant work experience, or a few computing courses plus some work experience. A student from a traditional computer science and engineering program may be deficient in some of the undergraduate core material, while the student with work experience will probably have a large number of deficiencies to overcome, although work experience may be substituted for some of the courses.

Educational Issues

In this section of the paper, a number of issues related to software engineering education are discussed. The issues include: undergraduate versus graduate education in software engineering, the structure of a graduate program in software engineering, maintaining a balance between fundamentals and techniques, providing realism in an educational environment, teaching of pervasive concepts, teaching of management and communication skills, the lack of qualified faculty and adequate textual materials, political considerations, and continuing education of practicing software engineers. Some of these issues have been discussed previously in references 3 and 4.

1. Undergraduate versus Graduate Education in Software Engineering

A continuing and unresolved controversy surrounds the issue of the level at which software engineering skills should be taught. On the one hand, many individuals who will become practicing software engineers will be Bachelor's level graduates, and proponents of undergraduate education in software engineering cite the success of electrical engineering educators in training electronic designers at the Bachelor's level. Although the undergraduate electrical engineer may not grasp the significance of various topics in his training at the time, it is argued, he will

soon appreciate the need for particular skills during his apprenticeship in industry. Opponents of undergraduate education in software engineering cite the need for maturity and experience as prerequisites to the study of software engineering. The opponents of undergraduate software engineering argue that software engineering is not a traditional engineering discipline in which concepts and job assignments can be nearly compartmentalized. Instead, a software engineer is a generalist who is concerned with computer science and engineering concepts and software engineering techniques, and is, in addition, highly competent in management and communication skills. The acquisition of these skills, argue the opponents, requires motivation and an understanding of the need for software engineering skills that will not be found in undergraduates.

Recognizing that there is no single solution to this controversy, the Software Engineering Curricula Subcommittee will follow the approach outlined in the previous section; namely, development of an undergraduate core of material that can be augmented into a full blown undergraduate program, and that can also be used as the undergraduate preparation for a Master's program in software engineering.

2. Structuring a Professional M.S. Program in Software Engineering

Determining the overall structure of a Master's program in software engineering is a difficult issue. One option is to require all courses of all students, with no flexibility in the program. The advantage of this approach is that all students are exposed to the same complete body of knowledge, as in professional programs such as law and medicine. The major disadvantage of this option is the long duration of the program, which would be a minimum of two years, assuming the student has satisfied all prerequisites before entering the program. A second option is to structure the program as a core of material, plus optional specializations. Thus, a student might specialize in software design, or validation, or project management. This provides the opportunity for in-depth specialization, and results in a shorter duration program, but presents the difficulty of establishing a small core of material that will provide the breadth of skills needed by a professional software engineer.

3. Balancing Fundamentals and Techniques

Maintaining a proper balance between fundamentals and techniques is a major issue in technological education. A primary goal of the university is to provide an education of lasting value by teaching basic concepts and fundamental principles which will form the basis for a satisfying and productive career. On the other hand, employers expect that new employees will possess sufficient practical skills to contribute to the mission of the organization without an extensive apprenticeship. Both theory and technique are essential components of a software engineering curriculum; however, there is a very real danger that software engineering programs will overcompensate for the lack of attention to practicalities in traditional computer science programs. Because industry practices are not standardized, skills of localized or short duration applicability, skills that cannot be generalized or related to basic principles, and skills that require extended periods of practical experience to acquire must be the responsibility of employers. Thus, an appreciation for the value of precise specifications, for example, can only be gained by learning and using a particular notational scheme for stating specifications; however, the emphasis should be placed on the need for, and the benefits of, formal specifications rather than on the pecularities of the scheme utilized, because the employer will probably not be using the particular scheme learned in school.

4. Providing Realism in an Educational Environment

It is often difficult to convince students that the methods of software engineering are valuable, and indeed essential, to success in software development and maintenance. The value of many of the techniques of software engineering become obvious only on large scale and/or long duration projects which involve turnovers in personnel, changing requirements, changes in machine environments, etc. In addition, classroom exercises lack true accountability for the end product, students are at best involved in their project assignments on a part-time basis, and problems of cost estimation and budgeting are difficult to convey in an academic environment. Several techniques have been proposed to overcome some of these problems. For example, it is essential that students acquire some experience in programming team techniques by working on a semester long or year long team project. Teams can be assigned different pieces of the project (thus forcing them to interface with other teams), partially completed work can be traded between teams, specifications can be changed in the middle of the project, and team members can be rotated among teams. An interesting approach to providing realistic experience in the university environment is described by Horning and Wortman (5). In the Software Hut approach, competing teams build portions of a system and attempt to sell (for course points) their portion to other teams who need it to complete the project. Selling points include the quality of the specification and design documents, understandability of the code, clarity of the interfaces, etc.

5. Teaching Pervasive Concepts

Many traditional engineering values pervade a software engineering curriculum. Concepts such as creative problem solving, reliability, testability, maintainability, performance criteria, design documentation, economics, and other quality considerations cut across every topic in software engineering. The problem facing software engineering educators is to design courses of study that insure proper emphasis on these values. Many of these concepts can be introduced in an immigration course, but they will need constant reinforcement throughout the program of study.

6. Teaching Management and Communication Skills

Management and communication skills are crucial components of software engineering. The high degree of interaction among people and machines is one of the primary factors that distinguishes software engineering from more traditional engineering disciplines. Although it is often possible to decompose a software system into isolated segments, it is often the

case that design decisions within a low-level segment of a system will have major ramifications throughout the system. Also, changes in high level specifications often propagate to the lowest levels of implementation. Thus, management and communication skills are essential tools for controlling complexity in a programming project. The traditional approach to teaching management and communications is to require courses in Organization Behavior, Public Speaking, and Technical Writing, which are taught by the Business and English Departments. Such brief exposure is not sufficient training for a software engineer. Instead of (or perhaps in addition to) a course in general technical writing, the software engineering student needs to know how to write requirements specifications, users' manuals, and software maintenance reports. Accuracy, precision, consistency, and completeness of expression are extremely important attributes of technical communication in software engineering, and the student needs specialized training in these skills. Not only does the software engineer need to know about PERT charts and CPM, but also how to use those techniques within the context of software production and maintenance. Thus, it is recommended that special courses for software engineers be developed in the management and communications areas. The major issue is whether the limited resources of most universities will permit the development of these specialized courses, and whether there are faculty members with the interest and qualifications to develop such courses.

7. Lack of Qualified Faculty and Textual Materials

The field of software engineering is in it's infancy; we predict that eventually software engineering will occupy a position in relation to computer science that electrical engineering occupies with respect to physics. At this time, however, there are very few faculty members who have had substantial experience in designing, constructing, and maintaining software systems in a production environment. Similarly, textual materials are inadequate, although the situation is gradually improving. The differential between university and industry salaries makes it difficult to attract and retain pragmatically oreinted teachers. University/industry exchange programs for faculty, formal specification of curricula in software engineering, and establishment of formal programs in software engineering will alleviate some of these problems.

8. Political Considerations

Software engineering is a multidisciplinary field, ranging from computer science and engineering techniques to management and communications. In most universities, the computer science department is most likely to have the most faculty expertise for teaching software engineering. However, electrical engineering and computer engineering groups have an orientation to design issues, economic considerations, and quality control. In addition, information systems departments in business schools often possess expertise in software project management techniques, programming team dynamics, scheduling and budgeting, accountability, and product visibility. The political issue is that each department has a legitimate interest in software engineering, yet none of them may be willing to allocate (or let anyone else allocate) the resources required to support a field that each regards as a subcomponent of their discipline. Eventually, we predict, departments of software engineering will be established with interdisciplinary faculties, although the departments may carry names other than software engineering, again due to the political consideration of the college that the program resides in within the university.

9. Continuing Education

This paper is primarily concerned with the educational issues of undergraduate and graduate programs in software engineering. There is, in addition, a tremendous need for the continuing education of practicing software engineers. The popularity of short courses and tutorials in the field are indicators of widespread interest in software engineering. It is fair to say, however, that continuing education can only be effective if the techniques being taught conform to the student's company policies and standards. A common comment of students in short courses and tutorials is that the material covered is interesting but that it cannot be used in their programming environment. One of the most effective approaches to continuing education is a series of intensive in-house training sessions that are incorporated into the implementation of company practices.

Conclusion

This paper has reviewed the curricular efforts of the Software Engineering Curricula Subcommittee, and several major issues in software engineering education have been discussed. Currently, the committee is preparing detailed description of the course material in a format similar to that of the Model Curriculum Report (2). The proposed curricula will be distributed to a review committee of 25 to 50 persons. Based on their comments, a final draft will be prepared by January 1979, and the report will be published in the spring of 1979.

This paper is quite obviously a report on work in progress. Several difficult issues are yet to be resolved by the committee. In curriculum design, as in most human endeavors, there is no single correct approach, but only the differing opinions of well informed, well intentioned individuals. We welcome your comments and constructive criticism of this work, as well as your opinions concerning the issues discussed in the paper.

References

1. P. Naur, B. Randell, and J.N. Buxton (ed.), Software Engineering: Concepts and Techniques, Petrocelli/Charter, New York, 1976.
2. Committee Report, A Curriculum in Computer Science and Engineering, IEEE Catalog No. EH0199-8, January 1977.
3. P. Freeman and A.I. Wasserman, "A Proposed Curriculum for Software Engineering Education", in Proc. 3rd Intl. Conf. on Software Engineering, IEEE Catalog No. 78CH1317-7C, May 1978.
4. M. Shaw, "Making Software Engineering Issues Real to Undergraduates", in Software Engineering Education: Needs and Objectives, Springer-Verlag, New York, 1976.

5. J.J. Horning and D.B. Wortman, "Software Hut: A Computer Program Engineering Project in the Form of a Game". in IEEE Transactions on Software Engineering, Vol. SE-3, No.4, July 1977.

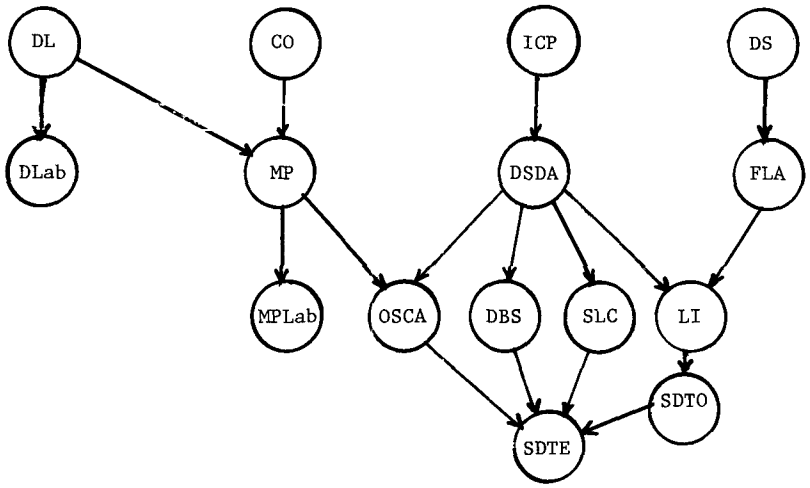

Figure 1. Structure of the Proposed Undergraduate Core