Abstract – An impediment to advancing the state of the software engineering discipline is the inadequate preparation of students entering the profession from undergraduate computing programs. This paper reports on the IEEE-CS/ACM project and the guidance it will provide for developing undergraduate software engineering programs. The organization of the project, its development principles and methods, and the deliverables of the project are enumerated. In particular, the paper outlines the organization and content of the proposed knowledge base for undergraduate software engineering education.

Index Terms – curriculum guidelines, software engineering undergraduate education

BACKGROUND

The Software Engineering Volume is part of the IEEE-CS/ACM Computing Curricula effort that began in the fall of 1998 when a group of dedicated individuals from the IEEE Computer Society and the ACM were appointed by the two societies to begin work on what was then called Computing Curriculum 2001. This Steering Committee came to the conclusion that the area of computing had gone beyond the boundaries of computer science and that one volume would not effectively contain all of the newer areas of computing that had been developed. It was decided to split the Computing Curriculum effort into several volumes: namely, Computer Science, Computer Engineering, Software Engineering, and Information Systems volumes. This Steering Committee proceeded to work on the Computer Science Volume while other groups were appointed to develop the other volumes.

Foundation work for the Software Engineering Volume began in 1998 with the Software Engineering Education Project (SWEEP) which produced a draft set of accreditation guidelines for software engineering published in IEEE Computer, April 1998 issue. In the fall of 2001, SWEEP was replaced by the Steering Committee for the Computing Curricula Software Engineering Volume (CCSE). Initial Steering Committee work included the development of a process plan, a set of guiding principles, a set of curriculum outcomes, and a set of templates to guide the detailing of what knowledge a graduate of a software engineering undergraduate program should possess.

GOALS AND GUIDING PRINCIPLES

The Steering Committee has articulated the following principles to guide CCSE work:
1. Computing is a broad field that extends well beyond the boundaries of any one computing discipline.
2. Software Engineering draws its foundations from a wide variety of disciplines.
3. The rapid evolution and the professional nature of software engineering require an ongoing review of the corresponding curriculum.
4. Development of a software engineering curriculum must be sensitive to changes in technology, new developments in pedagogy, and the importance of lifelong learning.
5. CCSE must go beyond knowledge elements to offer significant guidance in terms of individual curriculum components.
6. CCSE must support the identification of the fundamental skills and knowledge that all software engineering graduates must possess.
7. Guidance on software engineering curricula must be based on an appropriate definition of software engineering knowledge.
8. CCSE must strive to be international in scope.
9. The development of CCSE must be broadly based.
10. CCSE must include exposure to aspects of professional practice as an integral component of the undergraduate curriculum.
11. CCSE must include discussions of strategies and tactics for implementation, along with high-level recommendations.

As a first step in SE curriculum guidance, the following set of outcomes for an undergraduate curriculum in software engineering was developed. These outcomes were subsequently used during the construction of the software engineering education knowledge.

Graduates of an undergraduate SE program must be able to:
1. Work as part of a team to develop and deliver executable artifacts.
2. Understand the process of determining client needs and translating them to software requirements.
3. Reconcile conflicting objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations.
4. Design appropriate solutions in one or more application domains using engineering approaches that integrate ethical, social, legal, and economic concerns.
5. Understand and be able to apply current theories, models, and techniques that provide a basis for software design, development, implementation and verification.
6. Negotiate, work effectively, provide leadership where necessary, and communicate well with stakeholders in a typical software development environment.
7. Learn new models, techniques, and technologies as they emerge.

There are several hallmarks of the CCSE effort: the survey of prior software engineering education efforts [1-3], the public review process, and the inclusion of as many professionals from as many countries as possible in the Steering Committee and among our volunteers.

The Steering Committee has regularly sought input from the software engineering community. Workshops have been conducted at the 2002 and 2003 Conference on Software Engineering Education and Training, the 2002 and 2003 International Conference on Software Engineering and the 2002 STEP (Software Technology & Engineering Practice) meeting. Steering Committee members have participated in several meetings of the Working Group on Software Engineering Education and Training, where review and discussion of CCSE issues and artifacts has resulted in highly valuable contributions to the Volume’s contents.

The Steering Committee contains members whose mission is to guide the construction and detailing of the educational knowledge areas, guide the partitioning of these topics into a variety of academic classification schemes and implementations, and oversee the structure and content of the Volume. Other members serve as representatives to the views and perspectives of related professional groups: namely, the ACM, the ACM’s software engineering Special Interest Group, the two-year and community colleges subgroup of the ACM Educational Board, the IEEE Technical Committee on Software Engineering, the Australian Computer Society, the British Computer Society, and the Information Processing Society of Japan. In its entirety, the membership of the Steering Committee represents the countries of Australia, Canada, Israel, Japan, the United Kingdom, and the United States. The Steering Committee also seeks guidance from an advisory board.

International volunteers, software engineering educators and practicing software engineers were actively recruited to support our commitment to international representation in the CCSE project. Volunteers from Australia, Brazil, Canada, France, Germany, Hong Kong, Israel, Italy, Japan, Netherlands, New Zealand, Singapore, Spain, Taiwan, Thailand, Turkey, United Kingdom, United States, and Yugoslavia were and still remain involved as the Volume has developed. Despite the significant challenge of maintaining an international focus, an extensive effort has been made to do so. This contribution is most apparent in the areas of curriculum models.

**DEVELOPMENT PROCESS**

The CCSE Volume development process has proceeded in two distinct, overlapping stages: specify the core knowledge and prepare curriculum guidelines. These two stages were conducted using two groups of volunteers: an Education Knowledge Area group and a Pedagogy Focus group. The Education Knowledge Area group was responsible for defining and documenting a software engineering body of knowledge appropriate for guiding the development of undergraduate software engineering curricula. The Pedagogy Focus group was responsible for using the software engineering education knowledge to formulate guidance for pedagogy, and course and curriculum design to support undergraduate software engineering degree programs.

**Education Knowledge Area Group Work**

Work began in earnest on the volume in the spring of 2002 with the assignment of Education Knowledge Area volunteers to develop the body of Software Engineering Education Knowledge (SEEK). The volunteers were given an initial set of education knowledge areas, each with a short description, and were charged to define the units and topics for each knowledge area. This initial set was formed using past software engineering education efforts as a guide. An education knowledge area represents a particular sub-discipline of software engineering that is generally recognized as a significant part of the body of software engineering knowledge that an undergraduate should know. Each area is broken down into smaller divisions called units, representing individual thematic modules within an area, and topics, representing the subject matter contained within a unit. The initial work of the volunteers was incorporated in a preliminary draft of the SEEK and this document was used during the SEEK workshop held in June 2002 (sponsored by NSF DUE 02-29149). This workshop brought together Education Knowledge Area group members, Steering Committee members, leaders in software engineering education, and select Pedagogy Focus group members to evolve its contents.

The artifacts from the workshop were subsequently refined by the CCSE Steering Committee. A selected review of the resulting SEEK document was performed by a set of internationally recognized software engineering experts. Their evaluations/comments were used by the Steering Committee to refine and improve the volume's content.
Committee to produce the first draft version of the SEEK, which was released for public review in August 2002.

When the review window terminated in early October 2002, the Steering Committee had received approximately forty reviews. Each evaluation was coupled with a written response from the Steering Committee that included committee action and justification. After posting the second version of the SEEK in December 2002, another round of reviews was solicited until the beginning of March 2003. These evaluations along with the contributions of the Working Group on Software Engineering Education and Training were instrumental in sharpening the contents of the second version of the SEEK to best match the Pedagogy Focus group’s curriculum guidelines. The final version of the SEEK was released in April 2003.

The ten Software Engineering Education Knowledge areas, a designated number of contact hours for each area, and a short description of each area follow.

- **Computing Essentials** (172) Computing essentials includes the computer science foundations that support the design and construction of software products. This area also includes knowledge about the transformation of a design into an implementation, the tools used during this process, and formal construction methods.

- **Mathematical and Engineering Fundamentals** (89) The mathematical and engineering fundamentals of software engineering provide theoretical and scientific underpinnings for the construction of software products with desired attributes. These fundamentals support describing software engineering products in a precise manner. They provide the mathematical foundations to model and facilitate reasoning about these products and their interrelations, as well as form the basis for a predictable design process. A central theme is engineering design: a decision-making process of an iterative nature, in which computing, mathematics, and engineering sciences are applied to deploy available resources efficiently to meet a stated objective.

- **Professional Practice** (35) Professional Practice is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner. The study of professional practices includes the areas of technical communication, group dynamics and psychology, and social and professional responsibilities.

- **Software Modeling and Analysis** (50) Modeling and analysis can be considered core concepts in any engineering discipline since they are essential to documenting and evaluating design decisions and alternatives. Modeling and analysis is first applied to the analysis, specification and validation of requirements. Requirements represent the real world needs of users, customers and other stakeholders affected by the system and the capabilities and opportunities afforded by software and computing technologies. The construction of requirements includes an analysis of the feasibility of the desired system, elicitation and analysis of stakeholders' needs, the creation of a precise description of what the system should and should not do along with any constraints on its operation and implementation, and the validation of this description or specification by the stakeholders.

- **Software Design** (48) Software Design is concerned with issues, techniques, strategies, representations, and patterns used to determine how to implement a component or a system. The design will conform to functional requirements within the constraints imposed by other requirements such as resource, performance, reliability, and security. This area also includes specification of internal interfaces among software components, architectural design, data design, user interface design, design tools, and the evaluation of design.

- **Software Verification and Validation** (42) Software verification and validation uses both static and dynamic techniques of system checking to ensure that the resulting program satisfies its specification and that the program as implemented meets the expectations of the stakeholders. Static techniques are concerned with the analysis and checking of system representations throughout all stages of the software life cycle while dynamic techniques only involve the implemented system.

- **Software Evolution** (10) Software evolution provides cost-effective mission support during pre- and post-delivery stages while maintaining acceptable and satisfactory behavior and validity of assumptions. Such support requires pre- and post-delivery activities for preparation of each of a succession of versions or upgrades (releases) that constitute the evolving system. These activities require planning, logistics, metrics support, regression testing (including test library management) and complexity control/reduction (including redundancy removal). The techniques used to support these activities include program comprehension, successor release planning, needs and changes identification, re-engineering, reverse engineering, maintenance review, migration and system replacement/retirement.

- **Software Process** (13) Software process is concerned with knowledge about the description of commonly used software life-cycle process models and the contents of institutional process standards; definition, implementation, measurement, management, change and improvement of software processes; and use of a defined process to perform the technical and managerial activities needed for software development and maintenance.

- **Software Quality** (16) Software quality is a pervasive concept that affects, and is affected by all aspects of software development, support, revision, and maintenance. It encompasses the quality of work
products developed and/or modified (both intermediate and deliverable work products) and the quality of the work processes used to develop and/or modify the work products. Quality work product attributes include usability, reliability, safety, security, maintainability, flexibility, efficiency, performance and availability.

- **Software Management** (19) Software management is concerned with knowledge about the planning, organization, and monitoring of all software life cycle phases. Management is critical to ensure that software development projects are appropriate to an organization, work in different organizational units is coordinated, software versions and configurations are maintained, resources are available when necessary, project work is divided appropriately, communication is facilitated, and progress is accurately charted.

As part of an undergraduate software engineering education, students should specialize in one or more areas. Within their specialty, students should learn material well beyond the core material specified above. They may either specialize in one or more of the ten knowledge areas listed above, or they may specialize in one or more of potential application areas. For each application area, students should obtain breadth in the related domain knowledge while they are obtaining a depth of knowledge about the design of a particular system. Students should also learn about the characteristics of typical products in these areas and how these characteristics influence a system's design and construction. Examples of potential application areas are network-centric systems, information systems, embedded, real-time systems, and highly secure systems.

**Pedagogy Focus Group Work**

In October 2002, the Pedagogy Focus group began work on producing the curricula recommendations using the SEEK as a foundation. A Pedagogy Focus group process and work plan was formed and group members began defining the pedagogy guidelines, curriculum models, international adaptation, and implementation environments. This information was later refined by the Steering Committee during February 2003. Reviews of this draft of the Pedagogy Chapter of the Volume took place during the 2003 Conference on Software Engineering Education & Training and the 2003 International Conference on Software Engineering. A current draft of the pedagogy work contains the following information:

- Guidelines of software engineering curriculum design and delivery
- Proposed curricula which includes curriculum models based on modes of delivery
- Sample courses outlining what topics of the SEEK a particular course includes.
- Curriculum patterns

- International adaptation
- Classes of skills and problems that students should master, in addition to learning the knowledge in the SEEK
- Adaptation to alternative educational environments; e.g. two-year colleges

The curriculum models presented were developed using the SEEK, the Computer Science Volume (CCCS), and a survey of existing bachelors degree programs. A total of 32 programs from North America, Europe, and Australia were identified and characterized to aid in this work. A key technique to developing the models rested on identifying which SEEK topics would be covered by reusing existing CCCS courses. The remaining SEEK material was distributed into software engineering courses, using the existing programs as a guide. The resulting models represent implementations in a variety of institutional environments. It is certainly clear that all institutional environments will not be illustrated; however, the curriculum patterns were constructed to facilitate developing specialized programs. Such developmental tools are essential to support the ease of use of the Volume.

**CURRENT STATUS**

The current timetable calls for the first draft of the CCSE Volume to be released in the May/June time frame. Solicitations for review of the Volume will be made and all are encouraged to participate. The Steering Committee is particularly interested in comments pertaining to the Volume’s effectiveness in creating or extending software engineering programs or tracks. All CCSE Steering Committee artifacts and evaluations can be found on our web site [4].

**REFERENCES**


