Computing Curricula -- Software Engineering Volume

First Draft of the
Software Engineering Education Knowledge (SEEK)
August 28, 2002

Edited by
Ann E.K. Sobel
CCSE Knowledge Area Chair

Table of Contents

Objectives and Guiding Principles of CCSE

In the fall of 1998, the Educational Activities Board of the IEEE Computer Society and the ACM Education Board appointed representatives to a joint task force whose mission was to perform a major review of curriculum guidelines for undergraduate programs in computing. This activity, named Computing Curricula, and their corresponding final reports, which are listed as volumes II-V for the areas of Computer Science, Computer Engineering, Software Engineering, and Information Systems, are in varying stages of completion. The effort to create the software engineering volume is referred to as Computing Curricula Software Engineering (CCSE).

The CCSE steering committee is under the guidance and direction of both the IEEE Computer Society and the Association for Computing Machinery (see Appendix A for membership). 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 Australian Computer Society, the British Computer Society, and the Information Processing Society of Japan. As demonstration of the steering committee's commitment to generate an international curriculum, several international representatives also serve as members. 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 industrial advisory board.

CCSE Principles

The steering committee has articulated the following principles to guide our work:

1. *Computing is a broad field that extends well beyond the boundaries of any one computing discipline*. CCSE concentrates on the knowledge and pedagogy associated with a software engineering curriculum. Where appropriate, it will share or overlap with material contained in other Computing Curriculum reports and will offer guidance on its incorporation into other disciplines.

2. *Software Engineering draws its foundations from a wide variety of disciplines*. Undergraduate study of software engineering relies on many areas in computer science for its theoretical and conceptual foundations, but it also requires students to utilize concepts from a variety of other fields, such as mathematics, engineering and project management. All software engineering students must learn to integrate theory and practice, to recognize the importance of abstraction and modeling, to be able to acquire special domain knowledge beyond the computing discipline for the purposes of supporting software development in specific domains of application, and to appreciate the value of good engineering design.

3. *The rapid evolution and the professional nature of software engineering require an ongoing review of the corresponding curriculum.* The professional associations in this discipline must establish an ongoing review process that allows individual components of the curriculum recommendations to be updated on a recurring basis. Also, because of the special professional

responsibilities of engineers to the public, it is important that the curriculum guidance support and promote effective external assessment and accreditation of software engineering programs.

4. *Development of a software engineering curriculum must be sensitive to changes in technology, new developments in pedagogy, and the importance of lifelong learning*. In a field that evolves as rapidly as software engineering, educational institutions must adopt explicit strategies for responding to change. Institutions, for example, must recognize the importance of remaining abreast of progress in both technology and pedagogy, subject to the constraints of available resources. Software engineering education, moreover, must seek to prepare students for lifelong learning that will enable them to move beyond today's technology to meet the challenges of the future.

5. *CCSE must go beyond knowledge elements to offer significant guidance in terms of individual curriculum components*. The CCSE models should assemble the knowledge elements into reasonable, easily implemented learning units. Articulating a set of well-defined models will make it easier for institutions to share pedagogical strategies and tools. It will also provide a framework for publishers who provide the textbooks and other materials.

6. *CCSE must support the identification of the fundamental skills and knowledge that all software engineering graduates must possess.* Where appropriate, CCSE must help to define the common themes of the discipline and ensure that all undergraduate program recommendations include this material.

7. *Guidance on software engineering curricula must be based on an appropriate definition of software engineering knowledge*. The description of this knowledge should be concise, appropriate for undergraduate education, and it should use the work of previous studies on the software engineering body of knowledge. A core set of required topics, from this description, must be specified for all undergraduate software engineering degrees. The core should have broad acceptance by the software engineering education community. Coverage of the core will start with the introductory courses, extend throughout the curriculum, and be supplemented by additional courses that may vary by institution, degree program, or individual student.

8. *CCSE must strive to be international in scope.* Despite the fact that curricular requirements differ from country to country, CCSE is intended to be useful to computing educators throughout the world. Where appropriate, every effort is being made to ensure that the curriculum recommendations are sensitive to national and cultural differences so that they will be widely applicable throughout the world. The involvement by national computing societies and volunteers from all countries will be actively sought and welcomed.

9. *The development of CCSE must be broadly based*. To be successful, the process of creating software engineering education recommendations must include participation from the many perspectives represented by software engineering educators and by industry, commerce, and government professionals.

10. *CCSE must include exposure to aspects of professional practice as an integral component of the undergraduate curriculum*. The education of all software engineering students must include student experiences with the professional practice of software engineering. The professional practice of software engineering encompasses a wide range of issues and activities including problem solving, management, ethical and legal concerns, written and oral communication, working as part of a team, and remaining current in a rapidly changing discipline.

11. *CCSE must include discussions of strategies and tactics for implementation, along with high-level recommendations*. Although it is important for CCSE to articulate a broad vision of software engineering education, the success of any curriculum depends heavily on implementation details. CCSE must provide institutions with advice on the practical concerns of setting up a curriculum.

Curriculum Outcomes

As a first step in SE curriculum guidance the steering committee has developed the following set of outcomes for an undergraduate curriculum in software engineering:

Graduates of an undergraduate SE program must be able to:

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

Process of Determining the SEEK

The development model chosen for determining CCSE was based on the model used to construct the Computer Science Volume (CCCS). Development of the CCSE volume has been divided into two groups: an Education Knowledge Area Group and a Pedagogy Focus Group. The education knowledge area group is responsible for defining and documenting a software engineering education body of knowledge appropriate for guiding the development of undergraduate software engineering curricula (see Appendix B for list). This body of knowledge is called Software Engineering Education Knowledge or SEEK. The pedagogy focus group is responsible for using SEEK to formulate guidance for pedagogy as well as course and curriculum design to support undergraduate software engineering degree programs

The initial selection of the SEEK areas was based on the SoftWare Engineering Body Of Knowledge (SWEBOK) knowledge areas and multiple discussions with dozens of SEEK area

volunteers. The SEEK area volunteers were divided into groups representing each individual SEEK area where each group contained roughly seven volunteers. These groups were assigned the task of providing the details of the units that compose a particular educational knowledge area and the further refinement of these units into topics. To facilitate their work, references to existing related software engineering body of knowledge efforts (e.g. SWEBOK, CSDP Exam, and SEI curriculum recommendations) and a set of templates for supporting the generation of units and topics were provided.

After the volunteer groups generated an initial draft of their individual education knowledge area details, the steering committee held a face-to-face forum that brought together education knowledge and pedagogy area volunteers to iterate over the individual drafts and generate an initial draft of the SEEK (see Appendix C for attendee list). This workshop held with this particular goal mirrored a similar overwhelmingly successful workshop held by CCCS at this very point in their development process. Once the content of the education knowledge areas stabilized, topics were identified to be core (labeled with the designator essential) or elective (labeled with either the designator desirable or the designator optional). Topics were also labeled with the Bloom's taxonomy's levels of educational objectives; namely, knowledge, comprehension, or application. These three levels of learning were chosen from Bloom's taxonomy since they represent what knowledge may be reasonably learned during an undergraduate education.

The workshop resulted in a complete internal draft of SEEK. The steering committee then arranged for a review of the internal draft by selected experts in the field, the advisory industrial council, and the knowledge area volunteers (see Appendix D for list). After this review was complete, the steering committee studied all reviewer comments and used them to revise the internal draft version of the SEEK. This work resulted in a public draft version of the SEEK. The steering committee has made this version of the SEEK available to the public and is soliciting reviews of it by those interested in undergraduate software engineering education.

After the completion of public review of this document, the steering committee will use the review comments to produce a working version of SEEK. This version will be used to develop guidance for undergraduate pedagogy, courses, and curricula.

Knowledge Areas, Units, and Topics

Knowledge is a term used to describe the whole spectrum of content for the discipline: information, terminology, artifacts, data, roles, methods, models, procedures, techniques, practices, processes, and literature. The SEEK is organized hierarchically into three levels. The highest level of the hierarchy is the education knowledge **area,** representing 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. Knowledge areas are high-level structural elements used for organizing, classifying, and describing software engineering knowledge. Each area is identified by an abbreviation, such as PRO for professional practices and is represented in this document with the color orange. Each area is broken down into smaller divisions called **units,** which represent individual thematic modules within an area. Adding a two or three letter suffix to the area identifies each unit; as an example, PRO.com is a unit on communication skills. Units are represented in this document with the color yellow. Each unit is further subdivided into a set of **topics,** which are the lowest level of the hierarchy. Topics are represented with either the color teal or white.

Core Material

In determining the SEEK, the steering committee recognizes that software engineering, as a discipline, is relatively young in its maturation and common agreement on definition of an education body of knowledge is in an evolution stage. The SEEK developed and presented in this document is based on a variety of previous studies and commentaries on the recommended content for the discipline (see previous section). It was specially designed to support the development of undergraduate software engineering curricula, and therefore, does not include all the knowledge that would exist in a more generalized body of knowledge representation. The steering committee has therefore sought to define a minimal **core** consisting of the essential material that professionals teaching software engineering agree is necessary for anyone to obtain an undergraduate degree in this field. By insisting on a broad consensus in the definition of the core, the steering committee hopes to keep the core as small as possible, giving institutions the freedom to tailor the elective components of the curriculum in ways that meet their individual needs. Material offered as part of an undergraduate program that falls outside the core is considered to be **elective.** Core topics are represented with the color teal and elective topics are represented with no color (white).

The following points should be emphasized to clarify the relationship between the SEEK and the steering committee's ultimate goal of providing undergraduate software engineering curriculum recommendations.

- *The core is not a complete curriculum.* Because the core is defined as minimal, it does not, by itself, constitute a complete undergraduate curriculum. Every undergraduate program must include additional elective units from the body of knowledge, although this document does not define what those units will be.

- *Core units are not necessarily limited to a set of introductory courses taken early in the undergraduate curriculum.* Although many of the units defined as core are indeed introductory, there are also some core units that clearly must be covered only after students have developed significant background in the field. For example, topics in such areas as project management, requirements elicitation, and abstract high-level modeling may require knowledge and sophistication that lower division students do not possess. Similarly, introductory courses may include elective units alongside the coverage of core material. The designation *core* simply means *required* and says nothing about the level of the course in which it appears.

Unit of Time

The SEEK must define a metric that establishes a standard of measurement in order to judge the actual amount of time required to cover a particular unit. Choosing such a metric was quite difficult for the steering committee because no standard measure is recognized throughout the world. For consistency with the earlier curriculum reports, namely the other related computing curricula volumes to this effort, the task force has chosen to express time in **hours**. An hour corresponds to the actual in-class time required to present the material in a traditional lecture-oriented format (referred to in this document as contact hours). To dispel any potential confusion, however, it is important to underscore the following observations about the use of lecture hours as a measure:

- *The steering committee does not seek to endorse the lecture format.* Even though we have used a metric which has its roots in a classical, lecture-oriented format, the steering committee believes that there are other styles—particular given recent improvements in educational technology—that can be at least as effective. For some of these styles, the notion of hours may be difficult to apply. Even so, the time specifications should at least serve as a comparative measure, in the sense that a 5-hour unit will presumably take roughly five times as much time to cover as a 1-hour unit, independent of the teaching style.

- *The hours specified do not include time spent outside of class.* The time assigned to a unit does not include the instructor's preparation time or the time students spend outside of class. As a general guideline, the amount of out-of-class work is approximately three times the in-class time. Thus, a unit that is listed as requiring 3 hours will typically entail a total of 12 hours (3 in class and 9 outside).

- *The hours listed for a unit represent a minimum level of coverage.* The time measurements assigned for each unit should be interpreted as the *minimum* amount of time necessary to enable a student to perform the learning objectives for that unit. It is always appropriate to spend more time on a unit than the mandated minimum.

SE Education Knowledge Areas

In this section we describe the ten knowledge areas that make up the SEEK: Fundamentals (FND), Professional Practice (PRF), Software Requirements (REQ), Software Design (DES), Software Construction (CON), Software Verification & Validation (VAV), Software Evolution (EVL), Software Process (PRO), Software Quality (QUA), and Software Management (MGT). The knowledge areas do not include material about continuous mathematics or the natural sciences; the needs in these areas will be discussed in other parts of the CCSE volume. For each knowledge area, there is a short paragraph description and then a table that delineates the units and topics for that area. Each area's topics are listed with one of three attributes: the Bloom's taxonomy level (what capability should a graduate possess concerning the topic), whether a topic is essential (or desirable or optional) to the core, and the recommended core contact hours for the unit.

Bloom's attributes are specified using one of the letters k, c, or a, which represent:

- Knowledge (k) - remembering previously learned material. Test observation and recall of information, i.e., "bring to mind the appropriate information" (e.g. dates, events, places, knowledge of major ideas, mastery of subject matter).

- Comprehension (c) - understanding information and ability to grasp meaning of material presented. For example, translate knowledge to a new context, interpret facts, compare, contrast, order, group, infer causes, predict consequences, etc.

- Application (a) - ability to use learned material in new and concrete situations. For example, the use of information, methods, concepts, and theories to solve problems requiring the skills or knowledge presented.

There are some instances of designating a unit as having achieved the Bloom's taxonomy level of application. This designation on the unit has the interpretation that at least one of the topics of this unit must achieve the level of application.

A topic's relevance to the core is represented as follows:

- Essential (E) - the topic is part of the core.
- Desirable (D) - the topic is not part of the SEEK core, but it should be included in the core of a particular program if possible; otherwise, it should be considered as part of elective materials.
- Optional (O) - the topic should be considered as elective only.

Fundamentals

Description

The fundamentals of software engineering consist of the theoretical and scientific underpinnings describing attributes of the artifacts that software engineering produces, the mathematical foundations to model and facilitate reasoning about these artifacts and their interrelations, and the first principles that when applied produce predictable results; i.e., products with the desired attributes. A central theme is engineering design, a decision-making process of iterative nature, in which the "basic sciences", mathematics, and engineering sciences are applied to optimally convert resources to meet a stated objective.

Units and Topics

Reference		Blooms	Essential,	core contact
FND	**Fundamentals**	(k,c,a)	Desirable,	250
			Optional	
FND.mf	*Mathematical foundations+*			60
FND.mf.1	Functions, Relations and Sets (CCCS DS1)	a	E	
FND.mf.2	Basic Logic (prepositional and predicate) (CCCS DS2)	a	E	
FND.mf.3	Proof Techniques (direct, contradiction, inductive) (CCCS DS3)	a	E	
FND.mf.4	Basic Counting (CCCS DS4)	a	E	
FND.mf.5	Graphs and Trees (CCCS DS5)	a	E	
FND.mf.6	Discrete Probability (CCCS DS6)	a	E	
FND.mf.7	Finite State Machines, regular expressions	c	E	
FND.mf.8	Grammars	c	E	
FND.mf.9	Numerical precision, accuracy and errors	c	E	
FND.mf.10	Number Theory		D	
FND.mf.11	Algebraic Structures		O	
FND.cf	*Computing foundations**			140
FND.cf.1	Programming Fundamentals (CCCS PF1 to PF5) (control & data, typing, recursion)	a	E	
FNDcf.2	Algorithms, Data Structures/Representation (static & dynamic) and Complexity (AL 1 to AL 5)	a	E	
FND.cf.3	Problem solving techniques	a	E	
FND.cf.4	Abstraction – use and support for (encapsulation, hierarchy, etc)	a	E	
FND.cf.5	Computer organization (parts of CCCS AR 1 to AR 5)	c	E	
FND.cf.6	Basic concept of a system	c	E	
FND.cf.7	Basic user human factors (I/O, error messages, robustness)	c	E	

FND.cf.8	Basic developer human factors (comments, structure, readability)	c	E	
FND.cf.9	Programming language basics	c	E	
FND.cf.10	Operating system basics	c	E	
FND.cf.11	Database basics	c	E	
FND.cf.12	Network communication basics	c	E	
FND.ef	*Engineering foundations for software*			25
FND.ef.1	Empirical methods and experimental techniques (computer-related measuring techniques for CPU and memory usage)	c	E	
FND.ef.2	Statistical analysis (including simple hypothesis testing, estimating, regression, correlation etc.)	a	E	
FND.ef.3	Measuring individual's performance (e.g. PSP)	k	E	
FND.ef.4	Systems development (e.g. security, safety, performance, effects of scaling, feature interaction, etc.)	k	E	
FND.ef.5	Engineering design (e.g. formulation of problem, alternative solutions, feasibility, etc.)	c	E	
FND.ef.6	Engineering economics	k	E	
FND.ef.7	Engineering science for other engineering disciplines (strength of materials, digital system principles, logic design, fundamentals of thermodynamics, etc.)		O	
FND.md	*Modelling*			25
FND.md.1	Principles of modeling (level of abstraction, generalization, and composition)	a	E	
FND.md.2	Pre & Post conditions, invariants	c	E	
FND.md.3	Introduction to mathematical models and specification languages (Z, etc.)	c	E	
FND.md.4	Model checking and development tools	k	E	
FND.md.5	Properties of modeling languages	k	E	
FND.md.6	Syntax vs. semantics (understanding model representations)	k	E	
FND.md.7	Explicitness (make no assumptions, or state all assumptions)	k	E	

+Topics 1-6 correspond to Computer Science curriculum guidelines for discrete structures 1-6
* Computer Science curriculum guidelines CS1 and CS2 with the same breadth of computing but not in the same depth

Professional Practice

Description

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.

Units and Topics

Reference		Blooms	Essential,	core contact
PRF	**Professional Practice**	(k,c,a)	Desirable,	35
			Optional	
PRF.psy	*Group dynamics / psychology*			5

PRF.psy.1	Dynamics of working in teams/groups	c	E	
PRF.psy.2	Individual cognition (e.g. limits)	k	E	
PRF.psy.3	Cognitive problem complexity	k	E	
PRF.psy.4	Interacting with stakeholders	c	E	
PRF.psy.5	Dealing with uncertainty and ambiguity	k	E	
PRF.com	*Communications skills**			10
PRF.com.1	Reading, understanding and summarizing reading (e.g. source code, documentation)	a	E	
PRF.com.2	Writing (assignments, reports, evaluations, justifications, etc.)	a	E	
PRF.com.3	Team and group communication (both oral and written, email, etc.)	a	E	
PRF.com.4	Presentation skills	a	E	
PRF.pro	*Professionalism*			20
PRF.pro.1	Accreditation, certification, and licensing	k	E	
PRF.pro.2	Codes of ethics and professional conduct	c	E	
PRF.pro.3	Social, legal, historical, and professional issues and concerns	c	E	
PRF.pro.4	The nature of, and role of professional societies	k	E	
PRF.pro.5	The nature and role of software engineering standards	k	E	
PRF.pro.6	Engineering economics (economic impact aspects)	c	E	

* Specialized to the Software Engineering area.

Software Requirements

Description

Software requirements identify the purpose of a system and the contexts in which it will be used. Requirements act as the bridge between 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. These requirements must then be managed to consistently evolve with the resulting system during its lifetime.

Units and Topics

Reference		Bloom's	Essential,	core contact
REQ	**Requirements**	(k,c,a)	Desirable,	43
			Optional	
REQ.fd	*Requirements fundamentals*			6
REQ.fd.1	Definition of requirements (e.g. product, project, constraints, system boundary, external, internal, etc.)	c	E	
REQ.fd.2	Requirements process	k	E	
REQ.fd.3	Layers/levels of requirements (e.g. needs, goals, system requirements, software requirements, etc.)	c	E	
REQ.fd.4	Requirements characteristics (e.g. testable, non-ambiguous, consistent, correct, traceable, priority, etc.)	c	E	
REQ.fd.5	Special issues and considerations (e.g. prioritization and trade-off	c	E	

	analysis, risk, etc.)			
REQ.fd.6	Special perspectives (e.g. systems engineering, human-centered, etc.)		D	
REQ.fd.7	Wicked problems (e.g. ill-structured problems; problems with many solutions; etc.)		D	
REQ.fd.8	Interaction of requirements and architecture		D	
REQ.fd.9	COTS as a constraint		D	
REQ.el	*Eliciting requirements*			4
REQ.el.1	Elicitation Sources (e.g. stakeholders, domain experts, operational and organization environments, etc.)	c	E	
REQ.el.2	Elicitation Techniques (e.g. interviews, questionnaires/surveys, prototypes, use cases, observation, participatory techniques, etc.)	c	E	
REQ.el.3	Special techniques (e.g. ethnographic, knowledge elicitation, etc.)		O	
REQ.ma	*Requirements modelling and analysis*			15
REQ.ma.1	Modelling principles (e.g. decomposition, abstraction, projection/views, explicitness, use of formal approaches, etc.)	a	E	
REQ.ma.2	Information modelling (e.g. entity-relationship modelling, class diagrams, etc.)	a	E	
REQ.ma.3	Behavioral modelling (e.g. structured analysis, state diagrams, use case analysis, interaction diagrams, failure modes and effects analysis, fault tree analysis etc.)	a	E	
REQ.ma.4	Analyzing quality (non-functional) requirements (e.g. safety, security, usability, performance, etc.)	a	E	
REQ.ma.5	Enterprise modelling (e.g. business processes, organizations, goals, etc.)	k	E	
REQ.ma.6	Domain Modelling (e.g. domain engineering approaches, etc.)	k	E	
REQ.ma.7	Modelling embedded systems (e.g. real-time schedulability analysis, external interface analysis, etc.)		D	
REQ.ma.8	Requirements interaction analysis (e.g. feature interaction, house of quality, viewpoint analysis, etc.)		D	
REQ.ma.9	Analysis Patterns (e.g. problem frames, specification re-use, etc.)		O	
REQ.doc	*Requirements documentation*			9
REQ.doc.1	Requirements documentation basics (e.g. types, audience, structure, quality, attributes, standards, etc.)	k	E	
REQ.doc.2	Software requirements specification	a	E	
REQ.doc.3	Specification languages (e.g. structured English, formal languages such as Z, VDM, SCR, RSML)	k	E	
REQ.va	*Requirements validation*			6
REQ.va.1	Reviews and inspection	a	E	
REQ.va.2	Summative prototyping	k	E	
REQ.va.3	Formal analysis / model checking	k	E	
REQ.va.4	Acceptance test design	c	E	
REQ.va.5	Verifying quality attributes	c	E	
REQ.mgt	*Requirements management*			3
REQ.mgt.1	Change management	c	E	
REQ.mgt.2	Tracing	c	E	
REQ.mgt.3	Special management concers (e.g. consistency management, release planning, etc.)	k	E	

Software Design

Description

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.

Units and Topics

Reference		Blooms	Essential,	core contact
DES	**Design**	(k,c,a)	Desirable,	78
			Optional	
DES.con	*Software design concepts*			4
DES.con.1	Definition of design	c	E	
DES.con.2	Context of design within the software life cycle	k	E	
DES.con.3	Design principles (information hiding, cohesion and coupling)	c	E	
DES.con.4	Interactions between design and requirements	c	E	
DES.con.5	Design attributes (e.g. reliability, usability, performance, testability, fault tolerance, etc.)	k	E	
DES.con.6	Design trade-offs	k	E	
DES.con.7	Architectural styles and patterns	k	E	
DES.str	*Software design strategies*	a		12
DES.str.1	Function-oriented design	c	E	
DES.str.2	Object-oriented design	c	E	
DES.str.3	Data-structure centered design		O	
DES.str.4	Aspect oriented design		O	
DES.ar	*Architectural design*	a		15
DES.ar.1	Pipe and Filter	k	E	
DES.ar.2	Layered architectures	k	E	
DES.ar.3	Transaction Centered	k	E	
DES.ar.4	Peer-to-Peer	k	E	
DES.ar.5	Publish-subscribe and event-based	k	E	
DES.ar.6	Client-server, 3-tier, and containers	k	E	
DES.ar.7	Real-time and embedded	k	E	
DES.ar.8	Component-oriented, distributed objects, and middleware	k	E	
DES.ar.9	Architectural trade-offs	k	E	
DES.ar.10	Hardware issues in software architecture	k	E	
DES.ar.11	Requirements traceability in architecture	k	E	
DES.ar.12	Domain-specific architectures and product-lines	k	E	
DES.ui	*User interface design*			15
DES.ui.1	General UI design principles	a	E	
DES.ui.2	Use of modes, navigation	a	E	
DES.ui.3	Coding techniques and visual design	c	E	
DES.ui.4	Response time and feedback	a	E	

DES.ui.5	Design modalities (menu-driven, forms, question-answering,)	a	E	
DES.ui.6	Localization and internationalization	c	E	
DES.ui.7	User interface design methods	c	E	
DES.ui.8	Multi-media (e.g. I/O techniques, voice, natural language, web-page, sound, etc.)		D	
DES.ui.9	Metaphors and conceptual models		D	
DES.dd	*Detailed design*			20
DES.dd.1	Design methods: One selected method (e.g. component based)	a	E	
DES.dd.2	Other design methods	c	E	
DES.dd.3	Design patterns	a	E	
DES.dd.4	Component design	a	E	
DES.dd.5	Component interface design	a	E	
DES.dd.6	Algorithm design	a	E	
DES.dd.7	Data design	a	E	
DES.dd.8	Formal techniques for design	a	E	
DES.nst	*Design notations and support tools*	a		6
DES.nst.1	Architectural structure viewpoints and representations	c	E	
DES.nst.2	Functional structures (component diagrams)	c	E	
DES.nst.3	Object-oriented structures (class and object diagrams)	c	E	
DES.nst.4	Behaviour descriptions (e.g. state diagrams, Petri nets, pseudocode, data flow diagrams, etc.)	c	E	
DES.nst.5	Design support tools (e.g. architectural, static analysis, dynamic evaluation, etc.)	a	E	
DES.ev	*Evaluation*			6
DES.ev.1	Evaluation criteria (e.g. correctness, feasibility, soundness, etc.)	a	E	
DES.ev.2	Evaluation techniques (e.g. inspections, mathematically-based, static analysis, etc.)	a	E	
DES.ev.3	Design measurement and metrics	a	E	

Software Construction

Description

This area is concerned with knowledge about the development of the software components that are identified and described in the design documents. This area includes knowledge translation of a design into an implementation language, the development and execution of component tests, and the development and use of program documentation.

Units and Topics

Reference		Bloom's	Essential,	core contact
CON	**Construction**	(k,c,a)	Desirable,	46
			Optional	
CON.lan	*Language-oriented issues*			9
CON.lan.1	Programming style and naming conventions	a	E	
CON.lan.2	Programming idioms	a	E	
CON.lan.3	Parameterization and generics	a	E	

CON.lan.4	Assertions, design by contract, defensive programming	a	E	
CON.lan.5	Application oriented languages (e.g. scripting, visual, domain-specific, markup, macros, etc.)	a	E	
CON.tec	*Construction technologies*			25
CON.tec.1	Selection of data structures and algorithms	a	E	
CON.tec.2	API design and use	a	E	
CON.tec.3	Code reuse and libraries	a	E	
CON.tec.4	Object-oriented issues (e.g. polymorphism, dynamic binding etc.)	a	E	
CON.tec.5	Error handling, exception handling and fault tolerance	a	E	
CON.tec.6	State-based and table driven construction techniques	a	E	
CON.tec.7	Run-time configuration and internationalization	a	E	
CON.tec.8	Grammar-based input processing (parsing)	a	E	
CON.tec.9	Concurrency primitives (e.g. semaphores, monitors, etc.)	a	E	
CON.tec.10	Middleware (components and containers)	c	E	
CON.tec.11	Distributed software construction methods	a	E	
CON.tec.12	Constructing heterogeneous (hardware and software) systems; hardware-software codesign	c	E	
CON.tec.13	Platform standards (Posix etc.)		D	
CON.tec.14	Hot-spot analysis and performance tuning		D	
CON.tl	*Software Construction Tools*	a		2
CON.tl.1	Development environments	a	E	
CON.tl.2	GUI builders	c	E	
CON.tl.3	Unit testing tools	c	E	
CON.tl.4	Profiling, performance analysis and slicing tools		D	
CON.fm	*Formal construction methods*			10
CON.fm.1	Application of abstract machines (e.g. SDL, Paisley, etc.)	k	E	
CON.fm.2	Application of specification languages and methods (e.g. ASM, B, CSP, VDM, Z)	a	E	
CON.fm.3	Automatic generation of code	k	E	
CON.fm.4	Program derivation	c	E	
CON.fm.5	Algorithm and program analysis	c	E	
CON.fm.6	Mapping of a specification to different implementations	k	E	
CON.fm.7	Refinement	c	E	
CON.fm.8	Proofs of correctness		D	

Software Verification and Validation

Description

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.

Units and Topics

Reference		Bloom's	Essential,	core contact
VAV	**Verification and Validation**	(k,c,a)	Desirable,	46

			Optional	
VAV.fnd	*V&V terminology and foundations*			5
VAV.fnd.1	Objectives and constraints of V&V	k	E	
VAV.fnd.2	Planning the V&V effort	k	E	
VAV.fnd.3	Documenting V&V strategy, including tests and other artifacts	a	E	
VAV.fnd.4	Reliability metrics	k	E	
VAV.fnd.5	V&V involvement at different points in the lifecycle		D	
VAV.rev	*Reviews*			6
VAV.rev.1	Desk checking	a	E	
VAV.rev.2	Walkthroughs	a	E	
VAV.rev.3	Inspections	a	E	
VAV.tst	*Testing*			25
VAV.tst.1	Unit testing	a	E	
VAV.tst.2	Exception handling (writing test cases to trigger exception handling; designing good handling)	a	E	
VAV.tst.3	Coverage analysis (e.g. statement, branch, basis path, multi--condition, dataflow, etc.)	a	E	
VAV.tst.4	Black-box functional testing techniques	a	E	
VAV.tst.5	Developing test cases based on use cases and/or customer stories	a	E	
VAV.tst.6	Operational profile-based testing	k	E	
VAV.tst.7	System and acceptance testing	a	E	
VAV.tst.8	Testing across quality attributes (e.g. usability, security, compatibility, accessibility, etc.)	k	E	
VAV.tst.9	Regression Testing	c	E	
VAV.tst.10	Testing tools	a	E	
VAV.tst.11	Test-first programming		D	
VAV.tst.12	Deployment process		D	
VAV.uit	*User interface testing and evaluation*			6
VAV.uit.1	The variety of aspects of usefulness and usability	k	E	
VAV.uit.2	Measuring usability	c	E	
VAV.uit.3	Heuristic evaluation	a	E	
VAV.uit.4	Cognitive walkthroughs	c	E	
VAV.uit.5	User testing approaches (observation sessions etc.)	a	E	
VAV.uit.6	Web usability; testing techniques for web sites	c	E	
VAV.uit.7	Formal experiments to test hypotheses about specific UI controls		D	
VAV.par	*Problem analysis and reporting*			4
VAV.par.1	Analyzing failure reports	c	E	
VAV.par.2	Debugging/fault isolation techniques	a	E	
VAV.par.3	Defect analysis	k	E	
VAV.par.4	Problem tracking	c	E	

Software Evolution

Description

Software evolution is concerned with all activities that take place following the initial release of software. It includes techniques for handling evolution of software such as re-engineering, reverse engineering, program comprehension, process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration and retirement.

Units and Topics

Reference		Bloom's	Essential,	core contact
EVO	**Evolution**	(k,c,a)	Desirable,	9
			Optional	
EVO.pa	*Evolution processes and activities*			9
EVO.pa.1	Maintenance models (e.g. outsourcing, in-house, etc.)	k	E	
EVO.pa.2	Legacy system issues (e.g. wrappers, etc.)	k	E	
EVO.pa.3	Program comprehension and reverse engineering	k	E	
EVO.pa.4	Re-engineering	k	E	
EVO.pa.5	Impact analysis	k	E	
EVO.pa.6	Migration (technical and business)	k	E	
EVO.pa.7	Refactoring	k	E	
EVO.pa.8	Program transformations		D	
EVO.pa.9	Data reverse engineering		D	

Software Process

Description

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.

Units and Topics

Reference		Bloom's	Essential,	core contact
PRO	**Process**	(k,c,a)	Desirable,	16
			Optional	
PRO.con	*Process concepts*			3
PRO.con.1	Themes and terminology	k	E	
PRO.con.2	Software engineering process Infrastructure (e.g. personnel, tools, training, etc.)	k	E	
PRO.con.3	Modeling and specification of software processes	c	E	
PRO.con.3	Measurement and analysis	c	E	
PRO.con.4	Software engineering process improvement (small scale)	c	E	
PRO.con.5	Analysis and modeling of software process models		D	

PRO.imp	*Process Implementation*			13
PRO.imp.1	Levels of process definition (e.g. organization, project, team, individual, etc.)	k	E	
PRO.imp.2	Life cycle models (agile, heavyweight:waterfall, spiral, etc.)	c	E	
PRO.imp.3	Life cycle process models and standards (e.g., IEEE, ISO, CMM, SPICE, etc.)	c	E	
PRO.imp.4	Quality analysis and control (e.g. defect prevention, review processes, quality metrics, root cause analysis, etc.)	c	E	
PRO.imp.5	Individual software process (model, definition, measurement, analysis, improvement)	a	E	
PRO.imp.6	Team software process (model, definition, organization, measurement, analysis, improvement)	a	E	
PRO.imp.7	Process tailoring	k	E	

Software Quality

Description

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.

Units and Topics

Reference		Bloom's	Essential,	core contact
QUA	**Quality**	(k,c,a)	Desirable,	17
			Optional	
QUA.cc	*Software quality concepts and culture*			3
QUA.cc.1	Definitions of quality	k	E	
QUA.cc.2	Society's concern for quality	k	E	
QUA.cc.3	The costs and impacts of bad quality	k	E	
QUA.cc.4	A cost of quality model	c	E	
QUA.cc.5	Quality attributes for software	k	E	
QUA.cc.6	The dimensions of quality engineering	k	E	
QUA.cc.7	Roles of people, processes, methods, tools, and technology	k	E	
QUA.cc.8	Quality models and techniques (e.g. plan-do-check-act, TQM, etc.)		D	
QUA.std	*Software quality standards*			2
QUA.std.1	The ISO 9000 series	k	E	
QUA.std.2	ISO/IEEE Standard 12207: the "umbrella" standard	k	E	
QUA.std.3	Tailoring and adaptation of standards	k	E	
QUA.std.4	The ISO 10000 series		D	
QUA.std.5	IEEE software quality-related standards		D	
QUA.pro	*Software Quality Processes*			6
QUA.pro.1	Software quality models and metrics	c	E	
QUA.pro.2	Root cause analysis and defect prevention	c	E	
QUA.pro.3	Quality-related aspects of software process models	k	E	
QUA.pro.4	Introduction/overview of ISO 15504 and the SEI CMMs	k	E	
QUA.pro.5	Quality-related process areas of ISO 15504	k	E	

QUA.pro.6	Quality-related process areas of the SW-CMM and the CMMIs	k	E	
QUA.pro.7	The Baldridge Award criteria for software engineering		O	
QUA.pro.8	Quality aspects of other process models		O	
QUA.as	*Process and Product Assurance*			6
QUA.as.1	The nature of process and product assurance	k	E	
QUA.as.2	Distinctions between assurance and V&V	k	E	
QUA.as.3	Quality planning	a	E	
QUA.as.4	Organizing and reporting for process and product assurance	a	E	
QUA.as.5	Techniques of process and product assurance	c	E	

Software Management

Description

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.

Units and Topics

Reference		Bloom's	Essential,	core contact
MGT	**Software Management**	(k,c,a)	Desirable,	20
			Optional	
MGT.con	*Management concepts*			2
MGT.con.1	Classic management models	k	E	
MGT.con.2	Project management roles	k	E	
MGT.con.3	Enterprise/Organizational management structure	k	E	
MGT.con.4	Software management types (e.g. acquisition, project, development, maintenance, risk, etc.)	k	E	
MGT.pp	*Project planning*			10
MGT.pp.1	Evaluation and planning	c	E	
MGT.pp.2	Work breakdown structure	a	E	
MGT.pp.3	Task scheduling	a	E	
MGT.pp.4	Effort estimation	a	E	
MGT.pp.5	Resource allocation	c	E	
MGT.pp.6	Risk management	a	E	
MGT.per	*Project personnel and organization*			2
MGT.per.1	Organizational structures, positions, responsibilities, and authority	k	E	
MGT.per.2	Formal/informal communication	k	E	
MGT.per.3	Project staffing	k	E	
MGT.per.4	Personnel training, career development, and evaluation	k	E	
MGT.per.5	Meeting management	a	E	
MGT.per.6	Building and motivating teams	a	E	
MGT.per.7	Conflict resolution	a	E	

MGT.ctl	*Project control*			6
MGT.ctl.1	Change control	k	E	
MGT.ctl.2	Monitoring and reporting	k	E	
MGT.ctl.3	Measurement and analysis of results	c	E	
MGT.ctl.4	Correction and recovery	k	E	
MGT.ctl.5	Reward and discipline		O	
MGT.ctl.6	Standards of performance		O	
MGT.cm	*Software configuration management*			5
MGT.cm.1	Revision control	a	E	
MGT.cm.2	Release management	c	E	
MGT.cm.3	Tool support	c	E	
MGT.cm.4	Builds	c	E	
MGT.cm.5	Software configuration management processes	k	E	
MGT.cm.6	Maintenance issues	k	E	
MGT.cm.7	Distribution and backup		D	

Systems and Application Specialties

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 the application areas listed below. 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. Each application specialty listed below is elaborated with a list of related topics that are needed to support the application.

This list of application areas is not intended to be exhaustive but is designed to give guidance to those developing specialty curricula.

Specialties and Their Related Topics

Reference	
SAS	**System and Application Specialties**
SAS.net	*Network-centric systems*
SAS.net.1	Knowledge and skills in web-based technology
SAS.net.2	Depth in networking
SAS.net.3	Depth in security
SAS.inf	Information systems and data processing
SAS.inf.1	Depth in databases
SAS.inf.2	Depth in business administration
SAS.inf.3	Data warehousing
SAS.fin	*Financial and e-commerce systems*

SAS.fin.1	Accounting
SAS.fin.2	Finance
SAS.fin.3	Depth in security
SAS.sec	Fault tolerant and survivable systems
SAS.sur.1	Knowledge and skills with heterogeneous, distributed systems
SAS.sur.2	Depth in security
SAS.sur.3	Failure analysis
SAS.sur.4	Intrusion detection
SAS.sec	Highly secure systems
SAS.sec.1	Business issues related to security
SAS.sec.2	Security weaknesses and risks
SAS.sec.3	Cryptography, cryptanalysis, steganography, etc.
SAS.sec.4	Depth in networks
SAS.sfy	Safety critical systems
SAS.sfy.1	Depth in formal methods, proofs of correctness, etc.
SAS.sfy.2	Knowledge of control systems
SAS.emb	Embedded and real-time systems
SAS.emb.1	Hardware for embedded systems
SAS.emb.2	Language and tools for development
SAS.emb.3	Depth in timing issues
SAS.emb.3	Hardware verification
SAS.bio	Biomedical systems
SAS.bio.1	Biology and related sciences
SAS.bio.2	Related safety critical systems knowledge
SAS.sci	Scientific systems
SAS.sci.1	Depth in related science
SAS.sci.2	Depth in statistics
SAS.sci.3	Visualization and graphics
SAS.tel	Telecommunications systems
SAS.tel.1	Depth in signals, information theory, etc.
SAS.tel.2	Telephony and telecommunications protocols
SAS.av	Avionics and vehicular systems
SAS.av.1	Mechanical engineering concepts
SAS.av.2	Related safety critical systems knowledge
SAS.av.3	Related embedded and real-time systems knowledge
SAS.ind	Industrial process control systems
SAS.ind.1	Control systems
SAS.ind.2	Industrial engineering and other relevant areas of engineering
SAS.ind.3	Related embedded and real-time systems knowledge
SAS.mm	Multimedia, game and entertainment systems
SAS.mm.1	Visualization, haptics, and graphics
SAS.mm.2	Depth in user interface design

SAS.mm.3	Depth in networks
SAS.mob	Systems for small and mobile platforms
SAS.mob.1	Wireless technology
SAS.mob.2	Depth in user interfaces for small and mobile platforms
SAS.mob.3	Related embedded and real-time systems knowledge
SAS.mob.4	Related telecommunications systems knowledge

Appendix A

CCSE Steering Committee

Co-Chairs

Rich LeBlanc, ACM, Georgia Institute of Technology, U.S.
Susan Mengel, IEEE-CS, Texas Tech University, U.S.

Knowledge Area Chair

Ann Sobel, Miami University, U.S.

Pedagogy Focus Group Co-Chairs

Mordechai Ben-Menachem, Ben-Gurion University, Israel
Timothy C. Lethbridge, University of Ottawa, Canada

Co-Editors

Jorge L. Díaz-Herrera, Rochester Institute of Technology, U.S.
Thomas B. Hilburn, Embry-Riddle Aeronautical University, U.S.

Organizational Representatives

ACM: Andrew McGettrick, University of Strathclyde, U.K.
ACM SIGSOFT: Prem Devanbu, University of California at Davis, U.S.
ACM Two-Year College Committee: Elizabeth Hawthorne, Union County College, U.S.
Australian Computer Society: John Leaney, University of Technology Sydney, Australia
British Computer Society: David Budgen, Keele University, U.K.
Information Processing Society of Japan: Yoshihiro Matsumoto, Musashi Institute of Technology, Japan

Appendix B

Education Knowledge Area Volunteers

Tony (Anthony) Cowling
Peter Henderson
Doug Baldwin
Mark A. Ardis
Hossein Saiedian
Cynthia Cicalese
Mordechai Ben-Menachem
Niniek Angkasaputra
Peraphon Sophatsathit
Francis Pinheiro
Mohamed E. Fayad
David Dampier
Jocelyn Armarego
Jim Tomayko
Luisa Mich
Joseph E. Urban
Kai Chang
Mary Jane Willshire
Jason Chen
Vladan Devedzic
Oscar Dieste
Bimlesh Wadhwa

Mansour Zand
Stanislaw Jarzabek
Mike Lutz
Paul E. MacNeil
Moshe Krieger
Masao J. Matsumoto
Sira Vegas
Traian Muntean
Jianhan Zhu
Stephen C. Schwarm
Fawsy Bendeck
Mario Piattini
Arie van Deursen
Jens Jahnke
Roger Alexander
Natalia Juristo
Michael Oudshoorn
Emilia Mendes
Linda T. Taylor
Joel Henry
Yingxu Wang
Orit Hazzan
Valentina Plekhanova
Marta Lopez
Onur Demirors
Dick Fairley
James McDonald
Mike McCracken
Robert Burnett
Umit Karakas
Hareton Leung
Massood Towhidnejad
Dietmar Pfahl
Earl Beede
Jennifer S. Stuart
Jonathan D. Addelston
Ana Moreno
Mel Damodaran
Keith Olson
Richard Thayer
Yoshihiro Matsumoto
Atchutarao Killamsetty
Bill Hefley

Appendix C

CCSE Workshop Attendees

Rich LeBlanc
Susan Mengel
Ann Sobel
Timothy C. Lethbridge
Jorge L. Díaz-Herrera
Thomas B. Hilburn
Andrew McGettrick
David Budgen
Yoshihiro Matsumoto
Peter Henderson
Kai Chang
Traian Muntean
Earl Beede
Linda T. Taylor
Dick Fairley
Jenny Stuart
Steve Easterbrook
Bill Marion
Richard Upchurch
Tom Horton
Allen Parrish
Keith Olson
Mike McCracken
Frank Driscoll
Gideon Kornblum
Pierre Bourque
Haim Kilov
Cem Kaner
Frank H. Young
Barrie Thompson

Appendix D

Internal Reviewers