Comparison of SDD track in SCIS and ACM/IEEE Curriculum

Tasks:

1. Look into the ACM/IEEE curriculum for SE and determine if the core SE material can be covered in CEN 4010 - Software Engineering I and/or CEN 4021 – Software Engineering II.
2. Determine what courses students would need to take to keep the SDD track w.r.t the ACM/IEEE curriculum.
3. Determine how close the SDD track is to a full degree in SE.

Below are excerpts from the Computing Curricula for Science (CCCS) 2001 and the Software Engineering Education Knowledge (SEEK) as outlined in the SE 2004 ACM/IEEE curriculum report.

During the meeting I will present my findings.

Peter.

Computing Curricula 2001 Computer Science (December 15, 2001)

Sample approaches for the intermediate level

8.2.1 A traditional topic-based approach

As a sample implementation of this model, we propose the following set of courses:
CS210T. Algorithm Design and Analysis
CS220T. Computer Architecture
CS225T. Operating Systems
CS230T. Net-centric Computing
CS260T. Artificial Intelligence
CS270T. Databases
CS280T. Social and Professional Issues
CS290T. Software Development
CS490. Capstone Project

Figure 9-2. University model (US)

	semester 1	*semester 2*
year 1	CS101I. Programming Fundamentals Calculus I	CS102I. The Object-Oriented Paradigm CS115. Discrete Structures for Computer Science Calculus II
year 2	CS103I. Data Structures and Algorithms Science course I	CS120. Introduction to Computer Organization Science course II Probability and Statistics
year 3	CS210T. Algorithm Design and Analysis CS220T. Computer Architecture Advanced mathematics elective	CS225T. Operating Systems CS280T. Social and Professional Issues CS elective Undergraduate research project
year 4	CS230T. Net-centric Computing CS262T. Information and Knowledge Management CS290T. Software Development Undergraduate research project	CS490. Capstone Project CS elective CS elective

9.2 Advanced courses

Advanced computer science electives

Advanced courses serve three purposes, as follows:

1. Exposing the student to advanced material beyond the core
2. Demonstrating applications of fundamental concepts presented in the core courses
3. Providing students with a depth of knowledge in at least one subarea of computer science

As with the number of required mathematics courses, the exact number of electives in a given program will typically be a function of how much room is available in the curriculum, as well as college distribution requirements. However, the number of electives should be large enough to provide depth in at least one subarea of computer science. We propose a minimum of three advanced electives, while realizing that some schools may enlarge or decrease this number based on local conditions. We feel that three elective courses can provide sufficient opportunity for depth of study while keeping the overall program to a manageable size.

To ensure that students develop a reasonable level of depth in at least one subarea, it makes sense to require that a minimum of two out of three electives be chosen from a single area within the body of knowledge. The advanced courses are listed by area in Figure 9-1.

Part of Figure 9.1: Advanced courses by area

Software Engineering (SE)
CS390. Advanced Software Development
CS391. Software Engineering
CS392. Software Design
CS393. Software Engineering and Formal Specification
CS394. Empirical Software Engineering
CS395. Software Process Improvement
CS396. Component-Based Computing
CS397. Programming Environments
CS398. Safety-Critical Systems

Capstone project

The final component of this curriculum model is CS490, Capstone Project. This course provides students with opportunities to enhance skills that may not be easy to accomplish in the traditional classroom setting, such as working in teams, interacting with users, developing formal problem specifications, reviewing the research journals, building prototypes, scientific writing, and making oral presentations.

The most popular model for a capstone is a team-oriented, software engineering effort in which students design a solution to an information-based problem and work in teams to implement that solution. However, there is another model that might be more attractive to outstanding students who are thinking about graduate study and research, as opposed to private-sector employment. For these students, an alternative capstone format is a research experience that includes some original work, a review of the scientific literature, and an investigation of a proposed solution, followed by a scientific paper and/or an oral presentation of the results. It is important to remember that these are undergraduates and be realistic about the amount and quality of research expected. Even so, it may be more worthwhile to expose outstanding students to the challenges of research than to have them design and build yet another program.

Finally, each school must determine how long the capstone project will last. To truly get the most out of it (especially a research-based capstone) a year-long project is extremely beneficial. However, the resources available to a small department may constrain the project experience to a single semester.

Figure A-1. Computer science body of knowledge with core topics underlined

SE. Software Engineering (31 core hours)

SE1. Software design (8)
SE2. Using APIs (5)
SE3. Software tools and environments (3)
SE4. Software processes (2)
SE5. Software requirements and specifications (4)
SE6. Software validation (3)
SE7. Software evolution (3)
SE8. Software project management (3)
SE9. Component-based computing
SE10. Formal methods
SE11. Software reliability
SE12. Specialized systems development

Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building software systems that satisfy the requirements of users and customers. Software engineering is applicable to small, medium, and large-scale systems. It encompasses all phases of the life cycle of a software system. The life cycle includes requirement analysis and specification, design, construction, testing, and operation and maintenance. Software engineering employs engineering methods, processes, techniques, and measurement. It benefits from the use of tools for managing software development; analyzing and modeling software artifacts; assessing and controlling quality; and for ensuring a disciplined, controlled approach to software evolution and reuse. Software development, which can involve an individual developer or a team of developers, requires choosing the tools, methods, and approaches that are most applicable for a given development environment.

The elements of software engineering are applicable to the development of software in any computing application domain where professionalism, quality, schedule, and cost are important in producing a software system.

Software Engineering 2004 - Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering

Relationship to CCCS
The *CC2001 Computer Science* volume (CCCS) [ACM 2001] contains a set of recommendations for undergraduate programs in Computer Science. While undergraduate degrees in Software Engineering are different from degrees in Computer Science, the two have much in common, particularly at the introductory levels. We will refer to descriptions developed in CCCS when appropriate, and show how some of them can be adopted directly. This will be important for many institutions that offer both computer science and software engineering degrees.

Pattern N2S-1c - in a computer-science department

The pattern shown below is typical of a software engineering program that might be built in a computer science context. This is an adaptation of Pattern N2S-1, as shown above. Such programs may have evolved from computer science programs or may co-exist with computer science.

Year1		Year 2		Year 3		Year 4	
Sem 1A	Sem 1B	Sem 2A	Sem 2B	Sem 3A	Sem 3B	Sem 4A	Sem 4B
CS101	CS102	CS103	CS220	CS226	CS270T	SE400	SE400
Calc 1	*Calc 2*	CS106	SE A	MA271	SE D	SE F	*Tech elect*
NT181	CS105	SE201	SE212	SE C	SE E	*Tech elect*	*Tech elect*
Physics	*Any science*	NT272	*Linear Alg*	NT291	*Tech elect*	*Tech elect*	*Tech elect*
Gen ed	*Gen ed*	--	*Gen ed*	*Gen ed*	*Gen ed*	*Gen ed*	*Gen ed*

The following are titles and brief summaries of the courses in this package.

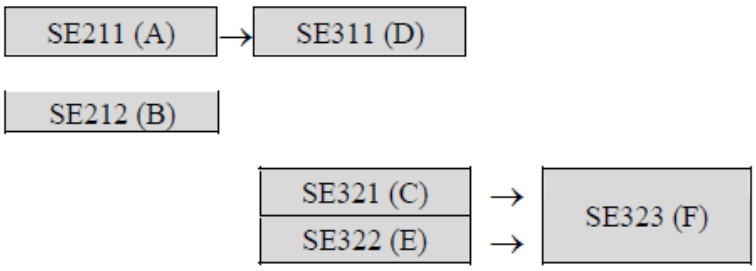
SE101 Introduction to Software Engineering and Computing
A first course in software engineering and computing for the software engineering student who has taken no prior computer science at the university level. Introduces fundamental programming concepts as well as basic concepts of software engineering.

SE102 Software Engineering and Computing II
A second course in software engineering, delving deeper into software engineering concepts, while continuing to introduce computer science fundamentals.

SE200 Software Engineering and Computing III
Continues a broad introduction to software engineering and computing concepts.

SE201 Introduction to Software Engineering
This is a central course, presenting the basic principles and concepts of software engineering and giving a firm foundation for many other courses described below. It gives broad coverage of the most important terminology and concepts in software engineering. Upon completing this course, students will be able to do basic modeling and design, particularly using UML. They will also have a basic understanding of requirements, software architecture, and testing.

6.3.1 Core Software Engineering Package I



SE211 Software Construction
Covers low-level design issues, including formal approaches.

SE212 Software Engineering Approach to Human Computer Interaction

Covers a wide variety of topics relating to designing and evaluating user interfaces, as well as some of the psychological background needed to understand people. This course is also found in Core Software Engineering Package II.

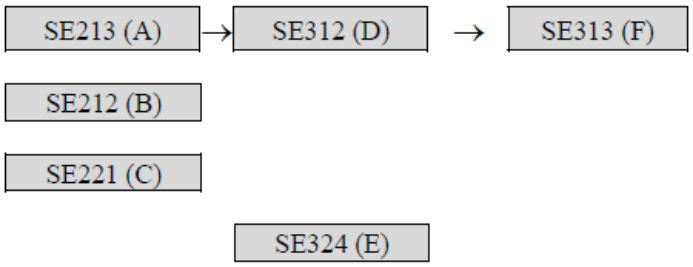
SE311 Software Design and Architecture
Advanced software design, particularly aspects relating to distributed systems and software architecture.

SE321 Software Quality Assurance and Testing
Broad coverage of software quality and testing.

SE322 Software Requirements Analysis
Broad coverage of software requirements, applied to a variety of types of software.

SE323 Software Project Management
In-depth course about project management. It is assumed that by the time students take this course, they will have a broad and deep understanding of other aspects of software engineering.

6.3.2 Core Software Engineering Package II



SE213 Design and Architecture of Large Software Systems
Modeling and design of large-scale, evolvable systems; managing and planning the development of such systems – including the discussion of configuration management and software architecture.

SE221 Software Testing
In-depth course on all aspects of testing, as well as other aspects of verification and validation, including specifying testable requirements, reviews, and product assurance.

SE312 Low-Level Design of Software
Techniques for low-level design and construction, including formal approaches. Detailed design for evolvability.

SE324 Software Process and Management
Software processes in general; requirements processes and management; evolution processes; quality processes; project personnel management; project planning.

SE313 Formal Methods in Software Engineering
Approaches to software design and construction that employ mathematics to achieve higher levels of quality. Mathematical foundations of formal methods; formal modeling; validation of formal models; formal design analysis; program transformations.

6.3.3 Software Engineering Capstone Project

As has been discussed in the guidelines presented in the last chapter, a capstone project course is essential in a software engineering degree program. The capstone course provides students with the opportunity to undertake a significant software engineering project, in which they will deepen their knowledge of many SEEK areas. It should cover a full-year (i.e. 80 lecture equivalent-hours). It covers a few hours of a variety of SEEK topics, since it is expected that students will learn some material on their own during this course, and will deepen their knowledge in several areas to the ‘a’ level of Bloom’s taxonomy.
SE400

SE400 Software Engineering Capstone Project
Provides students, working in groups, with a significant project experience in which they can integrate much of the material they have learned in their program, including matters relating to requirements, design, human factors, professionalism, and project management.