Specification for an Introductory Software Engineering Curriculum

submitted by the Working Group on Software Engineering and Training
June 2002

1. Introduction

This document contains an informal specification for the introductory part of a B.S. degree in Software Engineering (SE). The specification provides the framework for designing courses[1] and other learning activities that will help prepare students for more advanced study in software engineering. The curriculum is named "introductory" because it represents the beginning knowledge and practices that software engineering students must acquire in the first year or so to adequately prepare them for more advanced study. For example, we see the level of competency required in the introductory curriculum elements should be in the lower levels of Bloom's Taxonomy [2], at the "knowledge," "comprehension," and "application" levels.

The specification represents the work of the Working Group on Software Engineering and Training (WGSEET). In 1999, members of the WGSEET published an SEI technical report, titled *Guidelines for Software Engineering Education* [1]. The *Guidelines* addressed many of the curriculum design and pedagogy issues for software engineering that were covered for computer science in *Computing Curriculum 2001, Volume II, Computer Science* [4], but in a less expansive and detailed way. Also, since that time the *SWEBOK* [2] has evolved and matured. This document was influenced by the *Guidelines* effort and the SWEBOK. We do not mean to infer that the specification is complete, but believe it is a good starting point for those charged with developing detailed guidance for the introductory part of an undergraduate curriculum in software engineering.

Many of the members of the WGSEET are also working as volunteers in the CC2001 effort to establish guidance for developing a software engineering curriculum (CCSE). This document will be submitted to the Pedagogy Focus Area Group of the CCSE. One of the responsibilities of the Pedagogy Group is to design the introductory courses for a software engineering curriculum. We hope this document will assist in that activity.

[1] The word "course" is used to refer to a unit of instruction within a degree program (not the entire curriculum for a degree program). A student would typically take more than one course during an academic year.

2. Prerequisites

Students entering the introductory SE curriculum must be competent in pre-calculus mathematics. There are no prerequisites for programming skills or knowledge.

3. Curriculum Strategies and Pedagogical Issues

a. Students should engage in software engineering experiences they find both interesting and challenging.

b. The introductory curriculum should emphasize the teaching of problem solving and critical thinking.

c. Abstraction and modeling (a black box approach) should be used to teach software engineering principles.

d. Mathematics should be introduced as a software-modeling tool for small programs.

e. Software engineering principles should be introduced early in the curriculum.

f. Beginning students should be exposed to good software and programming practices, to illustrate the goals of professional software engineering. Likewise, examples of poorly developed software should be used to illustrate problems in software development.

g. Basic programming should be included as part of the introduction to software engineering. That is, software engineering principles and practices should be integrated into the teaching of programming in a high-level language.

h. Beginning students learn about and use a defined process for phased development of small programs.

i. Lectures/class discussions should be used should to teach concepts and principles, while and laboratories/tutorials should be used to teach software tools, processes, and practices.

j. Where possible the pedagogy should be “problem” driven, with multiple solutions per problem. That is, there is no “right” solution – rather, critical thinking should be applied to selecting the “most appropriate” solution.

k. The introductory curriculum should explore the nature of software engineering as a discipline (Science? Technology? Engineering? Is there a software science?).

l. The following is a framework for designing the first introductory course in a software engineering curriculum (SE 000):

i. Setting the Scene

What is good software? What is required to produce it? How does one determine the scope of a problem and scale it properly for solution? What are the rewards?

ii. Approaching Problems

Decomposition, Roadmap, Examples, Critical thinking

iii. Naming/Representation

Name-based solutions, Constraints, Spaces for names, Arrays of Names

iv. Procedural issues

Basic Building Blocks, Build Your Own Blocks, Information Hiding

4. Objectives

Upon completion of the introductory curriculum, students should be able to:

a. Describe the problems in software system development and evolution.

b. Discuss basic concepts and practices of software engineering, in the areas of requirements, design, construction, quality, and management.

c. Model, develop, and document a modest-sized, high-quality program in a high-level programming language.

d. Apply core computer science concepts, including basic data structures, to design and implement a modest software system.

e. Model a software artifact's properties, and argue its correctness using sound, but informal, reasoning.

f. Appreciate basic cooperative team skills, and communicate contributions and other issues with other members, in a clear and timely manner.

g. Describe the importance of ethical behavior and professional practice by a software engineer.

5. Specification and Implementation

In this section we elaborate on the material in the two previous sections (strategies and objectives), by adding detail and comment related to the implementation of the objectives. Although the comments differ in format and level of detail, they do represent an elaboration of the objectives, based on discussion within the WGSEET.

There were two overall cautionary comments voiced a number of times in the WGSEET discussions (paraphrased here): "We must remember that the content must be presented at an introductory/basic level and will require reinforcement and additional depth in subsequent parts of the curriculum"; and "It looks like too much material – we may be overloading students, setting them up for frustration and failure". Both of these comments seem to indicate at least some support within the group for the notion that, at minimum, the level of coverage of the topics that follow needs to be considered very carefully. Further consideration may also indicate that some of this material needs to be moved beyond the introductory courses.

a. Describe the problems in software system development and evolution.

   More specifically, students should understand problems in the following areas:
   > Product – size, complexity, invisibility
   > Communication – with teams; with users
   > Discipline – maturity, professional structure
   > People – shortage of professionals; cognitive limitations
   > Change – product change; technology change; application/social change; theory and practices change
   > Impact – potential for economic/physical change

b. Discuss basic concepts and practices of software engineering, in the areas of requirements, design, construction, quality, and management.

   More specifically, students should be able to
   > Understand and appreciate the importance of standards (e.g., IEEE Software Engineering Standards)
   > Describe the activities involved in requirements engineering (elicitation, analysis, modeling, specification, verification and validation).
   > Describe basic design concepts and principles (abstraction, modularity, information hiding, encapsulation), and various types and levels of design (architectural, data, user-interface, component, detailed).
   > Understand review, walkthrough and inspection to improve the quality of software artifacts.
   > Understand the cost of quality.
   > Understand techniques for planning and estimating.
   > Discuss the basic people issues involved in software engineering.

c. Model, develop, and document a modest-sized, high-quality program in a high-level programming language.

More specifically, students should be able to:

> Follow a simple defined process for product development to produce modest-sized software product that is accurate, readable, maintainable, fully documented, and does not crash.
> Produce a test suite to comprehensively challenge the program.
> Apply a quality process to analyze and improve beginning programming practices. Include the following elements:
  - the concept of quality
  - reviews, walkthroughs, inspections, testing
  - the general content of various lifecycle models

d. Apply core computer science concepts, including basic data structures, to design and implement a modest software system.

More specifically, students should be able to

> Apply critical thinking to the selection and use of basic data structures and algorithms and make design trade offs, taking into account complexity with appropriate measures of time and memory
> Select and apply static and dynamic based structures and common algorithms for these structures.
> Model a software artifact's properties, and argue its correctness using sound, but informal, reasoning.

More specifically, students should be able to:

> Use a notation (formal or informal) to represent a system's proprieties.
> Use logical reasoning to develop and understand loops and recursion.

e. Appreciate basic cooperative team skills, and communicate contributions and other issues with other members, in a clear and timely manner.

More specifically, students should be able to

> Describe the functions of a software team and the basic roles and responsibilities of team members, users, and customers.
- Define selected inter-personal skills and issues (e.g., listening, team dynamics, running meetings, personality differences) and explain why they are important to successful teams.

f. Describe the importance of ethical behavior and professional practice by a software engineer.

More specifically, students should be able to

> Identify the *Software Engineering Code of Ethics and Professional Practice* (ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices) as the key reference for a software engineering code of ethics
> Describe the consequences to society of unethical and unprofessional behavior by software engineers.
> Appreciate the need for professional conduct in the development of software.

> Appreciate the need for software engineers to engage in life-long learning.

6. References

[1] Bagert, D., et. al., Guidelines for Software Engineering Education, Version 1.0, CMU/SEI-99-TR-032, Software Engineering Institute, Carnegie Mellon University, 1999.

[2] Bloom, B.S. (Ed.) *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*, New York, Toronto: Longmans, Green, 1956.

[3] P. Bourque and R. Dupuis, eds., *Guide to the Software Engineering Body of Knowledge*, Stone Man Trial Version 0.95, http://www.swebok.org/, May, 2001.

[4] ACM/ IEEE-CS CC2001 Project, Computing Curricula 2001, *Computer Science Volume*, http://www.computer.org/education/cc2001/, December 2001.