The Evolutionary Changes of Undergraduate Software Engineering Curriculum

Barbara Bernal Thomas and Sheryl L. Duggins

Abstract

This article presents a concise history on the evolution of undergraduate software engineering curriculum over the decades and examines its apparent direction. Presented here are the cornerstones that shaped the foundations of what we as educators viewed as relevant and current over the years as a continuum of progress extending over two decades and paralleling the development of software engineering itself.

1. Introduction

Since the recent birth of software engineering at the 1968 NATO conference [NR69], the discipline has had unprecedented growth and academia has had a difficult time keeping up with the demand for skilled practitioners. Nearly a decade passed before the first framework for software engineering education was proposed. Another decade elapsed before the first model curriculum was designed and the software engineering degree programs began.

Marked by continual change, the last decade has seen steady progress in software engineering education (SEE). In a discipline that is this new, the question of what to teach is particularly difficult to answer; in an innovative field that is drastically changing as quickly as software engineering is, the question of curriculum takes on entirely new dimensions.

In 1997, the Department of Computer Science at Southern Polytechnic State University received permission to offer a Master of Science in Software Engineering (MSSE) degree [Mur99]. Designed as a vehicle for practitioners heavily involved in software development, the MSSE enabled meaningful academic credentials for future career opportunities and mobility. The MSSE at SPSU requires at least two years of prior experience in the software development/maintenance field.

In the second year SPSU offered the MSSE degree, we began a curriculum evaluation process that is still ongoing. During the course of the evaluation, we searched the literature for curriculum guidance while gathering material on other undergraduate software engineering programs currently offered. Through studying the software engineering education literature, we found reasoning within the evolution of undergraduate SE curriculum that shaped the foundations of what we as educators viewed as relevant and current at the time. For the purposes of this article, we are calling these points the cornerstones in the evolution of SE curriculum.

1 Southern Polytechnic State University, Computer Science Department, 1100 South Marietta Pkwy, Marietta, Georgia 30060, USA. Email: bthomas@spsu.edu Voice: (770) 528-4283 Fax: (770) 528-5511

2 Southern Polytechnic State University, Computer Science Department, 1100 South Marietta Pkwy, Marietta, Georgia 30060, USA. Email: sduggins@spsu.edu Voice: (770) 528-7401 Fax: (770) 528-5511
In some cases, they are simply representative of trends at that time. However, taken together, they form a continuum of progress extending over two decades and paralleling the development of software engineering itself.

Here we present the initial findings of our study to help other institutions in similar situations by providing a concise, although not totally complete, history of how software engineering curriculum has evolved over the decades and examination of where it appears to be heading.

2. Early History of Undergraduate Software Engineering Education

Proposed by Peter Freeman et. al. [FWF76], the earliest framework for SEE was identified as a set of criteria that any SE curricula must follow. The paper defined the following set of five content areas necessary for any SE degree: computer science, management science, communication, problem solving, and design. Emphasis was also placed on the need to incorporate both management and technical issues in software engineering.

The authors suggested the following criteria for any SE curricula.

1. It should be based on the above five content areas.
2. It must be flexible to keep up with developments in the field.
3. It should be based on computer science and viewed as “applied computer science.”
4. It must prepare students to push the boundaries of SE knowledge and not just apply knowledge.
5. It should be based on realistic and practical work.
6. It should provide for multiple implementations based on individual student needs.
7. It must build on the existing curricula as much as possible.

Revisiting SEE a decade later, Freeman [Fre87] reported that few, if any, efforts since his earlier paper had “strategically addressed the question of where SEE is or should be headed.” He further noted that in spite of the past ten years of development in software engineering, it is not an established part of the educational scene, nor was he aware of any master’s-level degree programs in SE at a major university. He did cite the workshops supported by the Software Engineering Institute(SEI) as a beginning for change. His final estimation of the proper foundation for SEE was his original ’76 criteria with a suggestion to add emphasis to design in SEE.

3. A Specification for Undergraduate Software Engineering

In February 1986, the Software Engineering Institute(SEI) had a workshop on Software Engineering Education. The participants revised an initial version of relevant subject areas for a SE degree. The resulting report published in May 1987 was titled Software Engineering Education, An Interim Report from the Software Engineering Institute [FGT87]. Presenting curriculum recommendations, this report was generally viewed as a specification for a professional Master of Software Engineering (MSE) degree and used as a guide in the creation of the Undergraduate Software Engineering degree.

The curriculum specification identified twenty content units. For each unit, topics were identified, aspects of those topics were listed, and educational objectives were given. The list of the twenty content areas follows.

1. The Software Engineering Process
2. Software Evolution
3. Software Generation
4. Software Maintenance
The interim report identified curriculum content without focusing on organizing those topics into courses. The following section describes the efforts in curriculum design that resulted in the first model SE curriculum, based on interim results.

**4. A Model for the First Curriculum in Undergraduate Software Engineering**

In February 1988 the SEI held a Curriculum Design Workshop [AF89] [Ard89] to design a curriculum for a SE degree based on the specification given in the interim report. The task was to partition the identified topics into courses. Based on the report, the designed curriculum would have 10 to 12 courses. Of these, six or seven would be core courses, three or four would be advanced electives, and the remainder would be used for project work.

The committee studied the twenty content units in the specification and identified five subject areas that naturally divided the topics: Systems Engineering, Software Design and Specification, Implementation, Verification and Validation, and Control and Management. Acknowledging that these five subject areas resembled the phases of the traditional waterfall life-cycle model, the committee concluded that the five areas were “legitimate as well as convenient partitions of the curriculum content.”

These twenty content units were partitioned into the five subject areas, with each unit assigned a relative size reflecting how many weeks would be required for coverage. Based on the size associated with the subject areas, the following six core courses were defined.

- Software Systems Engineering
- Specification of Software Systems
- Principles and Applications of Software Design
- Software Generation and Maintenance
- Software Verification and Validation
- Software Project Management

The committee noted that the courses were presented parallel to the phases in the order of a traditional waterfall life-cycle model: requirements, specification, design, implementation, and testing (plus project management). However they made no assumption that the curriculum would be always ordered or divided along that partition. Instead the partitioning needed to emphasize the different skills required of the students. They reinforced the idea that while schools will probably follow the life-cycle order, there are no prerequisite relationships among the required courses.
In addition to the core courses, the committee recommended that 30% of the program be devoted to project work with two additional appropriate semester-long project courses. They further suggested that approximately three electives be included in the curriculum, suggesting the following types of electives:

- Software engineering subjects
- Computer science subjects
- System engineering subjects.


Computing education as a discipline has grown to encompass various subdisciplines related to computing. In an effort to provide guidance for all the computer related fields, the Association of Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) jointly introduced the broad term “computing” to encompass all of the related disciplines. They published a broad set of curriculum guidelines, Computing Curriculum 1991 [Tuc90][Tuc91], that could be applied to any “computing” program.

Computing Curriculum 1991 identifies nine subject areas and three processes in computing. The areas are: algorithms and data structures, architecture methods, artificial intelligence and robotics, database and information retrieval, human-computer communication, numerical and symbolic computation, operating systems, programming languages, and software methodology and engineering. The three processes are theory, abstraction, and design. The software methodology and engineering area has the following five sub-areas: fundamental problem-solving concepts, the software development process, software requirements and specifications, software design and implementation, and verification and validation. Computing Curriculum 1991 does not present detailed curriculum design, which leaves issues of depth and breadth of coverage of the sub-areas still open. However, it did reinforce the prevalent SE life-cycle curriculum model since the areas were similar to the five subject areas in the first MSE model curriculum.

6. Experiential-Based Model

In 1991 the SEI proposed a report on SE education that identified 21 curriculum content units [For91]. The list was identical to the one given in the 1987 report [FGT87] and the 1989 report [AF89] with two additions. The new content unit of Professional Issues was added and the topics statistical testing concepts and techniques were added to the existing content unit Software Testing. The 1991 report identified 299 curriculum topics that were mapped onto the 21 curriculum units. The marked difference in the new report was a change in the delivery of the curriculum. This is noted in the discussion of the MSE curriculum structure that emphasizes a "spiral approach to education, in which material is presented several times in increasing depth. This approach is essential for a discipline such as software engineering, with many complex interrelationships among topics; no simple linear ordering of the material is possible [For91, p.27]."

A project experience component also had a noted emphasis in many SE programs as seen by the inclusion of capstone project courses, continuing projects, multiple course coordinated projects, cooperative programs with industry, university-based commercial software company, and design studios. This effort was an attempt to create a software engineering practicum in the curriculum. What is the role of work experience prior to entering a software engineering program? Some programs demand a couple of years of work experience in software engineering for entrance into the program. In 1988, SEI reported [AF88a] that students with previous SE work experience were more motivated to learn SE than students without relevant experience. Regardless of this, SEI chose not to make the work experience a prerequisite for entrance into the program and recognized the need for instructors to motivate students. In the 1991 report they discussed the design studio as a simulated work experience in the program.
7. The Carnegie Mellon University (CMU) New SE Model

Carnegie Mellon's SE program from 1989-1993 was based on the SEI model curriculum. Like most SE programs during that time period, their program was organized around the software life-cycle: it started with requirements elicitation and specification, and moved on to design, implementation, analysis and testing. By 1993, the general feeling at CMU was that they could improve their core curriculum. They began an evaluation of their curriculum that resulted in a redesign [GBJYW95].

They found there were advantages and disadvantages for utilizing the life-cycle model for their curriculum. Some advantages of designing the curriculum around the software life-cycle include: 1) the obvious fit – each topic has a place in software development; 2) it is a familiar path which many SE textbooks follow and support; and 3) each course is stand-alone and therefore convenient to part-time students. On the other hand, disadvantages cited include: 1) the emphasis on the waterfall model of software development; 2) the lack of emphasis of underlying techniques and principles of software development; and 3) the compartmentalization of each topic.

The result of this analysis was an emphasis not on the phases of development, but rather on the “cross-cutting principles of software development.” As Garlan et.al.[GBJYW95, p 69] state, “It emphasizes the underlying principles and techniques (such as the use of formal models and the application of good management principles) that can be applied in uniform ways to a broad spectrum of software development activities”. Thus the new core curriculum is organized around the following topics: modeling, methods of development, management, analysis, and architecture. Specifically, these topics resulted in the following five core courses.

- Models of Software Systems
- Methods of Software Development
- Management of Software Development
- Analysis of Software Artifacts
- Architectures of Software Systems

The MSE program has three organizational components: the Core Curriculum, various elective tracks, and the Software Development Studio. The Development Studio gives the students a chance to work in teams, with an external client, on a large-scale project under the direction of a faculty member. The Studio component continues over the entire duration of the program.

8. Software Engineering Body of Knowledge

The organizations of IEEE Computer Society and ACM have formed a committee named the Software Engineering Coordinating Committee (SWECC). This committee is sponsoring the Guide to Software Engineering Body of Knowledge (SWEBOK). SWEBOK is a three to four year project with three scheduled reports: Strawman, Stoneman and Ironman. Each report builds upon the previous one. Strawman has been completed and Stoneman is in progress.

The objectives of SWEBOK are to characterize the SE content, give free access to the report to all interested parties, and promote a consistent view of the body of knowledge in software engineering [BDAMTF99]. The scope of SWEBOK includes some knowledge of computer engineering, project management, quality engineering, mathematics, computer science, and other areas. It includes all software engineering specific knowledge [TF99]. The knowledge areas identified in the Stoneman report as of January 99 are:

1. Software Requirements Analysis
2. Software Design
3. Software Construction
4. Software Testing
These latest 10 Stoneman knowledge areas evolved and corresponded to the earlier 14 Strawman knowledge areas [BDAMTF99]. The original 14 Strawman knowledge areas were influenced by the ISO/IEC 12207 standard on Software Life-Cycle Processes. But there is not a one-to-one correspondence between the specific phases of the life-cycle and the knowledge areas. For example, in Stoneman the area of 7. Software Quality Analysis contains all quality related areas such as Quality Assurance, Verification and Validation, Dependability, and Software Reliability, which are not based on a phase of the life-cycle. Another example is 9. Software Engineering Infrastructure which contains the areas of Development Methods (including object-oriented, formal methods, and prototyping), and Software Development Environments (including explicit problems of reuse, standard designs and integration). These two examples do not represent a phase of the life-cycle and therefore extend the model beyond the influence of the ISO/IEC 12207 standard.

9. Future Work

This paper represents the first phase of a project for assessing undergraduate curricula. We are completing the information gathering stage and will proceed with analyzing the compiled material. This effort will lead to the development of an assessment model based on the historical perspective presented in this paper. The future work will propose possible mappings of current SE programs into our model. A tool for assessing recent SE undergraduates’ mastery of the specific knowledge areas will also be developed in conjunction with this project.

10. Conclusions

This article presented a concise history of how undergraduate software engineering curriculum has evolved. It examined the cornerstones that shaped the foundations of contemporary SE curriculum by tracing the progress of the last two decades. Efforts towards the SWEBOK were discussed in light of where SEE appears to be heading.

References


**Barbara Bernal Thomas**

Barbara Bernal Thomas is a full professor in the Computer Science Department at Southern Polytechnic State University (SPSU) for the last sixteen years. Prof. Thomas has taught the Introduction into Software Engineering and the Software Engineering Project courses since their inclusion into the undergraduate program. The areas of Software Engineering, User-Centered Design and Computer Graphics & Multimedia are the focus endeavors. Barbara is directly involved in the Usability Research Lab at SPSU. She has given numerous papers, tutorials and presentations locally and internationally on Software Engineering topics. Barbara is involved with computer educational support for local businesses in the Atlanta area. She does specialized software development as a consultant.

**Sheryl L. Duggins**

Sheryl L. Duggins is an associate professor in the Computer Science Department at SPSU where she teaches the graduate Software Engineering courses. Dr. Duggins has a Ph.D. from the University of Florida in Computer and Information Sciences and a M.S. from University of Missouri-Columbia in Computer Science. Her areas of research interest include object-oriented analysis and design, formal methods of software specification, software engineering, and software engineering education. Dr. Duggins’ consulting work includes the development and delivery of “Advanced Object-Oriented Analysis and Design Using the Booch Method” for Lockheed Martin. She was instrumental in the design and implementation of the Master of Science in Software Engineering (MSSE) degree at SPSU.