

# An Historical Investigation of Graduate Software Engineering Curriculum

Sheryl L. Duggins & Barbara Bernal Thomas  
Southern Polytechnic State University  
sduggins@spsu.edu & bthomas@spsu.edu

## Abstract

*This article presents a concise history of the evolution of graduate 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. The current impact of the SWECC and the SWEBOK project on software engineering curriculum is examined and the significance that licensing professional software engineers will have on what we teach is discussed.*

## 1. Introduction

Since the recent birth of software engineering at the 1968 NATO conference [11], 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 program 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 an effort to stay current, the Department of Computer Science at Southern Polytechnic State University conducted a curriculum evaluation study of our Master of Science in Software Engineering (MSSWE) degree. During the course of the evaluation, we searched the literature for curriculum guidance while gathering material on other graduate software engineering programs currently offered. Through studying the software engineering education literature, we found various points in the evolution of graduate SE curriculum that shaped the foundations of what 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. These cornerstones form a continuum of progress extending over two decades that parallels the development of software engineering itself.

One of the cornerstones that is of particular relevance today is the work done by the Software Engineering Coordinating Committee (SWECC) on the Software Engineering Body of Knowledge (SWEBOK) project. Their efforts are investigated along with the current impact of the SWECC and the SWEBOK project on software engineering curriculum. The significance that licensing professional software engineers will have on what we teach is also discussed.

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 examining where it appears to be heading. Admittedly, we were limited to exploring papers and program descriptions written in English or Chinese, and we acknowledge the limitations of the study. Furthermore, the graduate programs we examined were limited to Masters in Software Engineering, since our focus is on curriculum issues. Our hope is that this paper may serve both to give an overview of SE curricular issues as well as to jump-start the investigation for other schools considering adding a software engineering degree to their program.

## **2. Early History of Graduate Software Engineering Education**

Proposed by Peter Freeman et. al. [8], 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 [9] 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. While still affirming his initial foundation for SEE, he added that design must have an increased emphasis in SEE.

## **3. A Specification for Graduate Software Engineering**

In February 1986, the participants of SEI’s Software Engineering Education Workshop revised an initial version of relevant subject areas for a graduate SE degree. The resulting report published in May 1987 was titled Software Engineering Education, An Interim Report from the Software Engineering Institute [6]. Presenting curriculum recommendations, this report was generally viewed as a specification for a professional Master of Software Engineering (MSE) 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 are shown in Graph 1, Section 1987.

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 the interim report.

#### **4. A Model for the First Master's in Software Engineering**

In February 1988 the SEI held a Curriculum Design Workshop [2,3] to design a curriculum for an MSE 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"[2].

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, six core courses were defined as stated in Graph 1, Section 1988.

The committee noted that while the courses parallel the phases of a traditional waterfall life-cycle model: requirements, specification, design, implementation, and testing (plus project management), they did not use that as the basis for their design. Rather, the partitioning emphasized the different skills required of the students. They reinforced the idea that while schools would probably order the courses following the life-cycle chronology, 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 semester-long project courses being appropriate. They further suggested that approximately three electives be included in the curriculum, and recommended the following types of electives:

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

#### **5. Computing Curriculum 1991**

Computing education as a discipline has grown to encompass various sub-disciplines related to computing. In an effort to provide guidance for all 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 [16,17], 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 Software Engineering Institute proposed a report on graduate SE education that identified 21 curriculum content units [7]. The list was identical to those presented in the 1987 report [6] and in the 1989 report [2,3] with two additions. A new content unit titled Professional Issues was added and the topics Statistical Testing Concepts and Techniques were added to the existing Software Testing content unit. 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" [7, 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 companies, and design studios. This effort was an attempt to create a software engineering practicum in the curriculum. As an alternative to a practicum, some programs required one or two years of work experience in software engineering for entrance into the program. In 1988, SEI reported [2] 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 CMU MSE Model

Carnegie Mellon's MSE program from 1989-1993 was based on the SEI model curriculum. Like most graduate 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 [10].

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 conducive 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.[10, 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 five core courses shown in Graph 1, Section 1995.

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**

In 1993, the Joint IEEE Computer Society and ACM Steering Committee for the Establishment of Software Engineering as a Profession was created. In 1998, this committee was superseded by the Software Engineering Coordinating Committee (SWECC) and was established to act as a “permanent entity to foster the evolution of software engineering as a professional computing discipline”[13]. It was determined that achieving consensus by the profession on a core body of knowledge was crucial for the evolution of software engineering to a profession. With the approval of both IEEE-CS and ACM, SWECC set up the Guide to the Software Engineering Body of Knowledge (SWEBOK) project [13] to identify that subset of the body of knowledge that is “generally accepted.” The objectives of the SWEBOK project are to:

- Characterize the contents of the Software Engineering Body of Knowledge;
- Provide a topical access to the Software Engineering Body of Knowledge;
- Promote a consistent view of software engineering worldwide;
- Clarify the place of, and set the boundary of, software engineering with respect to other disciplines such as computer science, project management, electrical engineering and mathematics;
- Provide a foundation for curriculum and individual certification and licensing material.

SWEBOK is a four year project with three scheduled reports: Straw Man, Stone Man and Iron Man. Each report builds upon the previous one. Straw Man and Stone Man have been completed and Iron Man is currently under development. Achieving consensus by the profession has been facilitated by the open solicitation of reviews of the working reports [16].

The initial Straw Man report selected the Knowledge Areas (KAs) based on “recognized, public and verifiable sources of information” including “tables of contents of general software engineering textbooks, the curricula of undergraduate and graduate programs in software engineering and the admission criteria for graduate programs”[13]. The Straw Man KAs are shown in Graph 1, Section 1998 [13]. Potential knowledge areas were refined based on the ISO/IEC 12207 standard on Software Life-Cycle Processes. Thus, as in the earlier

cornerstones we have presented, the Straw Man knowledge areas still represent a life-cycle model.

The Straw Man report also identified the related disciplines of: [5]

1. Cognitive science and human factors,
2. Computer engineering,
3. Computer science,
4. Management and management science,
5. Mathematics,
6. Project management, and
7. Systems engineering.

The Stone Man report is divided in KA sections that include the knowledge area identification and specification including breakdowns, rationale, standards, and relevant ties to the seven related disciplines above. The knowledge areas identified in the Stone Man report are given in Graph 1, Section 2000 [14].

One major shift in direction that can be seen by examining the progression of the Stone Man documents is a gradual departure from the life-cycle model. There is a continuum of refinement of the knowledge areas away from partitioning based on phases of the life-cycle, to a new, continually evolving software engineering body of knowledge. It is evident that the Stone Man report extends the SE model beyond the influence of the ISO/IEC 12207 standard.

## **9. Licensing of Professional Software Engineers**

The reader may have noticed that the original question proposed in this paper of what to teach professional software engineers has not yet been addressed. The curriculum studies presented thus far have illustrated the evolution of software engineering curriculum over the last two decades. The distinction between software engineering and professional software engineering is still fuzzy, and the role that licensing will play is still under debate.

There has been a clear move towards establishing software engineering as a profession since the steering committee was formed in 1993; the issue has been elevated to a much higher level of prominence with the establishment of SWECC and the influence of SWEBOK. Many papers have been written that discuss what being a professional means (see [4] for a sample), and while most agree that SE is or should be considered a profession, the related issue of whether that implies a need for licensing is far from settled.

As specified initially, the SWEBOK project had five objectives. However, many believe the fifth objective: to “provide a foundation for curriculum and individual certification and licensing material” was given much more emphasis than originally intended. Concerned about the direction SWECC was moving, ACM established task forces to investigate the issue of licensing software engineers [1]. The study found an “explicit and intimate link” between the SWEBOK project and “the intent and expectation for software engineering licensing” [1]. Based on the study, the ACM Council decided in May 1999 that it could not support licensing of software engineers. ACM’s rationale was that current software engineering practice is too immature to warrant licensing, and that licensing would not provide assurances about software quality and reliability.

In May 2000, the ACM Council decided that the framework of a licensed professional engineer, which was originally developed for civil engineers, “does not match the professional industrial practice of software engineering. Such licensing practices would give false assurances of competence even if the body of knowledge were mature; and would preclude many of the most qualified software engineers from becoming licensed”[12]. Because SWECC became so closely linked with licensing of software engineers, the ACM Council decided to withdraw from SWECC.

The impact that this decision will have on software engineering curriculum is yet to be seen. The distinction between the software engineering body of knowledge and curriculum was discussed in the Straw Man report. It made a clear distinction between the two, stating there were clearly things a software engineer must know outside of software engineering. However, the body of knowledge identified in the project should form the core of software engineering curriculum. Thus it follows that if there is a move towards licensing software engineers under the rubric of the Professional Engineers Licensing structure and requirements, there would be a definite impact on the curriculum of software engineering programs. The examination for licensing professional engineers “would require examinations over subjects most software engineers neither study in their formal education nor need in order to practice software engineering” [1]. Specifically, in addition to computers and math, the fundamentals of engineering examination covers chemistry, ethics, statics, dynamics, electric circuits, and thermodynamics [15].

The SWECC represented a joint effort involving both IEEE-CS and ACM working together to develop a core body of software engineering knowledge and to foster the evolution of software engineering to a profession. The recent withdrawal of ACM from the SWECC due to differences in opinion on the emphasis being placed on licensing software engineers has abruptly changed the perceived value of the contributions of SWEBOK. As the assessment report states, “Overall, it is clear that the SWEBOK effort is structurally unable to satisfy any substantial set of the requirements we identified for bodies of knowledge in software engineering, independent of its specific content” [12]. Due to its significance to SEE, we view this latest development as another cornerstone in the evolution of software engineering curriculum.

## **10. The Significance Licensing Software Engineers has on Curriculum Design**

The Department of Computer Science at Southern Polytechnic State University has recently received approval for a Bachelor’s of Software Engineering (BSSwE) degree. The proposal sent to the Board of Regents included a preliminary curriculum for the new program, and we recently completed a curriculum evaluation study to refine the new degree before it was initially offered. This Fall our new BSSwE is being offered for the first time.

A large part of our evaluation and related discussions was significantly linked to the current status of the licensing issue. Since very few schools currently offer undergraduate software engineering programs, we want our program to be viewed as a model curriculum that is based on the latest developments in software engineering education. Thus we are actively watching to see if the trends in licensing SE’s as professional engineers, that is, by taking the PE examination common to all engineering fields, will continue after the ACM’s withdrawal from the SWECC, or if some new model will evolve. Our concern is that if the licensing trend continues, then we must produce students who will have the requisite background to pass the licensing exam. But on a pragmatic note, we already had to make difficult choices to keep the number of hours required for the BSSwE down to fit the

maximum allowed. There simply is no room in our curriculum to include the additional courses in traditional engineering (e.g., statics, dynamics, electric circuits, thermodynamics, etc.) without removing a number of core software engineering courses that cover the common body of knowledge necessary for software engineering.

The problem we at Southern Polytechnic State University are facing is one that software engineering educators must face together. Either we align our programs and ourselves with the SWECC's efforts towards licensing, or we follow the lead of the ACM and wait and see. Our BSSwE does not initially include the traditional engineering fundamentals, but we plan to advise our students on the licensing issues, and we are considering how we might be able to offer the engineering courses as an optional track, or perhaps as a certificate program.

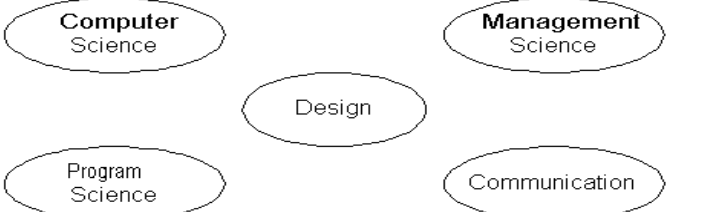
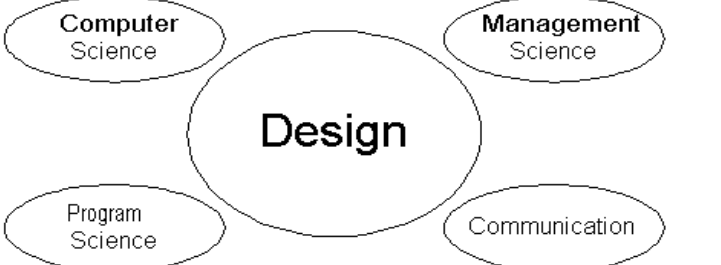
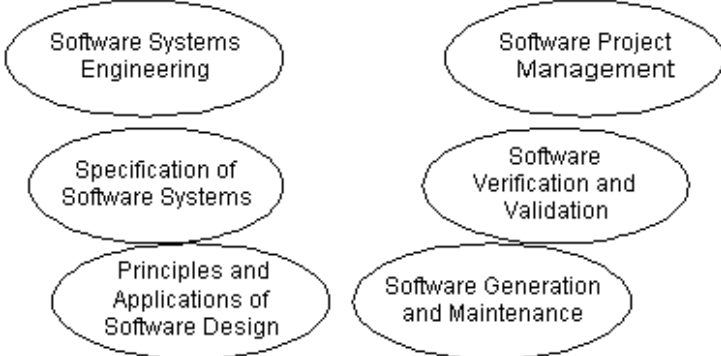
## 11. Conclusions

This article presented a concise history of how graduate 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 how they effect SEE. Issues related to software engineering as a profession and the licensing of software engineers were discussed and their effects on software engineering curriculum were explored.

## 12. References

- [1] "A Summary of the ACM Position on Software Engineering as a Licensed Engineering Profession" (July 17, 2000) [http://www.acm.org/serving/se\\_policy/selep\\_main.html](http://www.acm.org/serving/se_policy/selep_main.html)
- [2] Ardis, Mark and Ford, Gary (1989) 1989 SEI Report on Graduate Software Engineering Education. Technical Report CMU/SEI-89-TR-21, Software Engineering Institute.
- [3] Ardis, Mark and Ford, Gary (1989) "SEI Report on Graduate Software Engineering Education" in Proceedings of the Software Engineering Education Conference, edited by Gibbs, N., July 1989, Springer-Verlag.
- [4] Boehm, B., Brooks, F., Browne, J. Gray, J., Hawthorne, P., Kennedy, K., Parnas, D., & Wulf, W.A. Software Engineering & Licensing Position Papers. [http://www.acm.org/serving/se\\_polcy/papers.html](http://www.acm.org/serving/se_polcy/papers.html)
- [5] Dupuis, R., Bourque, P., Abran, A., Moore, J. W., & Tripp, L. (March 26, 1999) A Baseline for a list of Related Disciplines for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge, <http://www.swebok.org/>
- [6] Ford, G., Gibbs, N., and Tomayko, J. (1987) Software Engineering Education: An Interim Report from the Software Engineering Institute. Technical Report CMU/SEI-87-TR-8, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
- [7] Ford, Gary. 1991 SEI Report on Graduate Software Engineering Education, Technical Report CMU/SEI-91-TR-2, Software Engineering Institute, April 1991.
- [8] Freeman, Peter, Wasserman, A.I., and Fairley, R. E., (1976) "Essential Elements of Software Engineering Education", in Proceedings of the 2nd International Conference on Software Engineering, pp. 116-122.
- [9] Freeman, Peter (1987) "Essential Elements of Software Engineering Education Revisited." IEEE Transactions on Software Engineering, SE-13, pp. 1143-1148.
- [10] Garlan, D., Brown, A., Jackson, D., Tomayko, J., and Wing, J. (1995). "The CMU Master of Software Engineering Core Curriculum" in Proceedings of the 8th Software Engineering Education Conference, edited by Ibrahim, Rosalind, April 1995, Springer-Verlag, pp. 65-86.
- [11] Naur, P., and Randell, B., (1969). Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee, NATO.
- [12] Notkin, D., Gorlick, M., & Shaw, M. (May 2000) An Assessment of Software Engineering Body of Knowledge Efforts.
- [13] Guide to the Software Engineering Body of Knowledge - Straw Man Version - September 1998.
- [14] Guide to the Software Engineering Body of Knowledge – Stone Man Version Version 0.95 SWEBOK February 2000. <http://www.swebok.org/>
- [15] Tripp, L.L. (March 22, 1999) Professionalization of Software Engineering: Next Steps

- [16] Tripp, L. & Frailey, D. J. (Feb. 2, 1999) IEEE Computer Society and ACM Software Engineering Coordinating Committee (SWECC) Overview.
- [17] Tucker, Alan B., (Editor) et.al., Report of the ACM/IEEE-CS Joint Curriculum Task Force. <http://www.acm/education/curr91/homepage.html>

Date	Proposed																								
1976	Freeman																								
1987	Freeman																								
1987	SEI Workshop	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" data-bbox="527 1058 1263 1094" style="text-align: center;"><b>Curriculum Content</b></th> </tr> </thead> <tbody> <tr> <td data-bbox="527 1094 927 1129">The Software Engineering Process</td> <td data-bbox="927 1094 1263 1129">Software Operational Issues</td> </tr> <tr> <td data-bbox="527 1129 927 1165">Software Evolution</td> <td data-bbox="927 1129 1263 1165">Requirements Analysis</td> </tr> <tr> <td data-bbox="527 1165 927 1201">Software Generation</td> <td data-bbox="927 1165 1263 1201">Specification</td> </tr> <tr> <td data-bbox="527 1201 927 1236">Software Maintenance</td> <td data-bbox="927 1201 1263 1236">System Design</td> </tr> <tr> <td data-bbox="527 1236 927 1272">Technical Communication</td> <td data-bbox="927 1236 1263 1272">Software Design</td> </tr> <tr> <td data-bbox="527 1272 927 1308">Software Configuration Management</td> <td data-bbox="927 1272 1263 1308">Software Implementation</td> </tr> <tr> <td data-bbox="527 1308 927 1344">Software Quality Issues</td> <td data-bbox="927 1308 1263 1344">Software Testing</td> </tr> <tr> <td data-bbox="527 1344 927 1379">Software Quality Assurance</td> <td data-bbox="927 1344 1263 1379">System Integration</td> </tr> <tr> <td data-bbox="527 1379 927 1415">Software Project Organizational and Management Issues</td> <td data-bbox="927 1379 1263 1415">Embedded Real-time Systems</td> </tr> <tr> <td data-bbox="527 1415 927 1451">Software Project Economics</td> <td data-bbox="927 1415 1263 1451">Human Interfaces</td> </tr> </tbody> </table>		<b>Curriculum Content</b>		The Software Engineering Process	Software Operational Issues	Software Evolution	Requirements Analysis	Software Generation	Specification	Software Maintenance	System Design	Technical Communication	Software Design	Software Configuration Management	Software Implementation	Software Quality Issues	Software Testing	Software Quality Assurance	System Integration	Software Project Organizational and Management Issues	Embedded Real-time Systems	Software Project Economics	Human Interfaces
<b>Curriculum Content</b>																									
The Software Engineering Process	Software Operational Issues																								
Software Evolution	Requirements Analysis																								
Software Generation	Specification																								
Software Maintenance	System Design																								
Technical Communication	Software Design																								
Software Configuration Management	Software Implementation																								
Software Quality Issues	Software Testing																								
Software Quality Assurance	System Integration																								
Software Project Organizational and Management Issues	Embedded Real-time Systems																								
Software Project Economics	Human Interfaces																								
1988	SEI																								

1991	Computing Curr.	Subject Areas:		Processes:	
		Algorithms and Data Structures		Theory	
		Architecture Methods		Abstraction	
		Artificial Intelligence & Robotics		Design	
		Database & Information Retrieval			
		Human-Computer Communication			
		Numerical and Symbolic Computation			
		Operating Systems			
		Programming Languages			
Software Methodology and Engineering					
1995	CMU				
1998	Straw Man Knowledge Areas	Development Process		Improvement Process	
		Maintenance Process		Software Development Methods	
		Configuration Management		Software Development Environments	
		Quality Assurance		Measurement/Metrics	
		Verification and Validation		Software Reliability	
		Software Engineering Overview and Definition			
2000	Stone Man Knowledge Areas	Software Requirements		Software Configuration Management	
		Software Design		Software Engineering Management	
		Software Construction		Software Engineering Process	
		Software Engineering Testing		Software Engineering Tools and Methods	
		Software Maintenance		Software Quality	

Graph 1