A Comparison of Computer Science and Software Engineering Programmes in English Universities

Farid Meziane and Sunil Vadera
School of Computing, Science and Engineering
University of Salford,
Salford M5 4WT, UK
{f.meziane, s.vadera}@salford.ac.uk

Abstract

Recent years have seen much debate about the appropriate content of Software Engineering (SE) programs and how they relate to Computer Science (CS) programs, culminating in the distinguishing knowledge areas identified in the ACM/IEEE CS and SE curricula. Given these publications, a reasonable question to ask is: how do current SE programs differ from CS programs and to what extent do the differences reflect the characterizing features given in the ACM/IEEE curricula? This paper aims to answer these questions for SE programs offered in England. The content of a third of the SE programs in England are analyzed and summarized with respect to the knowledge areas of both the ACM/IEEE CS and SE curricula. The results reveal interesting features, such as intelligent systems is a more distinguishing feature between the CS and SE programs than the expected knowledge areas given in the SE curriculum. The main finding is that there are relatively few differences between existing SE and CS programs offered in England. The paper concludes with a discussion of the reasons for this situation and its likely implications.

1. Introduction

Software Engineering (SE) is regarded as new compared to other engineering disciplines. Indeed, it was first mentioned at a NASA conference organized in 1968 [9]. Since then, many methodologies, programming languages and architectures have been developed. Universities had to adapt traditional Computer Science (CS) programs to include concepts of SE to satisfy the demand for skilled practitioners. However, given the complexity and diversity of the SE discipline, designing, implementing and delivering SE programs is not a simple task and the content of software engineering education (SEE) is still debatable. Early frameworks for SEE [7,8] stressed the importance of flexibility in the design of these programs and the proposed contents areas included computer science, management sciences, communications, problem solving and design. SE programs were also viewed as "applied Computer Science" as opposed to "pure Computer Science" programs that are regarded as being more general and theoretical. Duggins and Thomas [6] give a good historical investigation of the SE curriculum.

The early SE programs did not live up to the expectations of many practitioners and an ever-demanding industry. There were debates on whether to classify SE as an engineering discipline, with strong views about the lack of "Engineering" in SE programs. Indeed, SE courses are embedded in general CS programs, which are not all focused on SE. Another debate was on the dissociation of SE programs from CS programs and the possible delivery of SE programs outside the traditional CS departments [10] The last few years have seen a lot of interest in the SEE from both industry and academia [5,11,12,17]. This interest is supported by many projects such as the Software Engineering Body of Knowledge Project (SWEBOK) [3] that attempts to define the knowledge required by a

Proceedings of the 17th Conference on Software Engineering Education and Training (CSEET'04)
1093-0175/04 $20.00 © 2004 **IEEE**

software engineer and the Computing Curriculum-Software Engineering project (SS-SE) [2] that includes recommendations and guidance on the content of SE programs. A key question for debate when making these recommendations has been the distinguishing features of SE programs as compared to CS programs. Having established the distinguishing features of SE programs, a natural follow up questions is: "How do existing CS and SE programs measure up against the distinguishing features?"

This paper aims to answer this question for Universities in England. The remainder of this paper is organized as follows. Section 2 lists the key features of CS and SE curricula as defined by the ACM/IEEE task force. Section 3, presents the results of a survey of the content of SE and CS programs in some English universities. The results of this research are discussed in the conclusion given in section 4.

2. Key Features of CS and SE

This section summarizes the key distinguishing features of CS and SE programs. The expected characteristics of CS curricula were established by the ACM/IEEE joint task force on computing curricula 2001 [1] which identified a set of 14 areas that together represent the body of knowledge for CS at undergraduate level. The content of the ACM/IEEE'2001 curriculum and its rationale are well documented and not repeated here, except to note that the key areas identified are: Discrete Structures, Programming Fundamentals, Algorithms and Complexity, Operating Systems, Net-Centric Computing, Programming Languages, Human Computer Interaction, Graphics and Visual Computing, Intelligent systems, Information Management, Social and Professional Issues, Software Engineering and Computational Science.

Discussion about the content of SE programs is more recent and probably worthy of some discussion in this paper in terms of process Parnas' [10] work has been influential. He defined three steps to follow to produce a SE programme. He first defines the possible tasks a software engineer is expected to perform, followed by definition of a body of knowledge required for the Software Engineer and then its implementation as a training program.

For the first step, Parnas [10] gave the following list of tasks a software engineer is expected to perform:

- Elicitate, determine the requirements for an application and record them in a precise, well organized and easily used documents.
- Participate in the design of software and determine how the different functions of the system will be implemented considering all software and hardware platforms.
- Analyze the performance of the design and ensure that the proposed system meets user requirements.
- Design the structure and architecture of the system and check its consistency, completeness and suitability for the intended application.
- Implement the document the software
- Integrate new software with existing software.
- Perform software testing
- Revise and enhance software systems, maintaining their integrity and keeping all documents complete and accurate.

Based on these tasks, Parnas argued that a software engineering program should comprise four categories of courses:

1. Basic courses taken by all engineering disciplines (G Courses): This include General Chemistry, Engineering Mathematics (4 courses), Calculus (2 courses), Introductory Mechanics, Engineering Design and Communications, Safety Training, Waves, Electricity and Magnetic Fields, Introductory programming for Engineers and Engineering Economics.
2. Courses for software engineers that provide an overview of basic engineering issues (E Courses): This include Introduction to the Structure and Properties of Engineering Material, Introduction to Dynamics and Control of Physical Systems, Digital System Principles and Logic Design for Software Engineers, Architectures of Computers and Multiprocessors, Introduction to Thermodynamics and Heat Transfer.
3. Courses on the mathematical foundations of SE (M courses): This includes Application of Mathematical Logic in Software Engineering, Applications of discrete Mathematics in Software Engineering, Statistical methods for Software Engineering.
4. Courses in Software design courses (S Courses). This is the core of the programs. 22 courses in total were suggested and this includes Programming, Software Design, Communication Skills, Structures and Algorithms, Human Computer Interaction, Distributed and Parallel Systems and various projects.

The IEEE/ACM task force on computing curricula went through a similar process in defining SE programmes. However, before defining the tasks required by a Software Engineer (they called this student outcomes) they started by defining the principles of a SE program. A total of eleven principles, such as diversity, change and internationalization were defined (see [2] pages 8-9 for more details). This was followed by the definition of Software Engineering Education Knowledge (SEEK). This was then further divided into knowledge area, units and topics. Later in the documents, some curriculum patterns were proposed. This process led to the following knowledge areas that are used in this paper as characterizing a software engineering program: Computing Essentials, Mathematical and Engineering Fundamentals, Professional Practice, Software Modeling and Analysis, Software Design, Software Verification and Validation, Software Evolution Software Process, Software Quality, Software Management and Systems and Application Specialties.

3. Comparison of SE and CS Curricula

Most English universities offer bachelor degrees in CS and SE that are of three or four years duration[1]. Four year versions in England usually consist of a third year that is an industrial placement year where students spend a year in the IT industry, gaining professional experience. Programs in the UK are structured on a level by level basis where progression to the next level is determined by success at a previous level. Modules at higher levels are usually more demanding, with their content expected to comply with the UK Qualifications Framework [13] that characterizes the nature of each level and award.

The following methodology was used to carry out the comparison. First, the Universities and Colleges Admissions System (UCAS) [15] was used to identify the 44 institutions that offer both a SE and a CS program. From these, the syllabus for a third of the programs was obtained, analyzed and mapped to the core topics of the CS and SE ACM/IEEE curricula

[1] Programs in Scotland are 4 years long and have not been part of the survey.

outlined in section 2 above. The key aim of the mapping was to assess the coverage of each topic. Ideally, we would have liked to assess content on an hourly basis. However, this level of detail is rarely available within institutions, let alone accurately specified for external purposes. Hence, the analysis focused on assessing the number of modules that contribute significantly towards the core topics[2]. Although this is a broader analysis than one based on hours, we believe it does give a representative view of the overall nature of the programs.

Tables 1 and 2 give the results of the mapping. Each table presents the number of modules that contribute significantly to a topic on a level by level basis. Table 1 gives the mapping of the CS and SE programs to the ACM/IEEE CS topics and Table 2 gives the mapping to the ACM/IEEE SE topics. Figures 1 and 2 contrast the results which are discussed in the next section.

Table 1: CS and SE Programs with regards to the ACM/IEEE CS Curriculum

Core Topics	CS Programmes				SE Programmes			
	Year 1	Year 2	Year 3	Total	Year 1	Year 2	Year 3	Total
1 Discrete Structures	13	0	0	13	10	1	1	12
2 Programming Fundamentals	13	2	1	16	10	4	0	14
3 Algorithms and Complexity	4	9	2	15	4	7	1	12
4 Architecture and Organization	11	3	2	16	9	3	1	13
5 Operating Systems	4	5	3	12	1	6	2	9
6 Net-Centric Computing	4	7	9	20	6	10	7	23
7 Programming Languages	0	5	7	12	0	8	6	14
8 Human-Computer Interaction	4	3	5	12	4	3	4	11
9 Graphics and Visual Comput.	0	3	5	8	1	2	4	7
10 Intelligent Systems	2	8	11	21	2	5	6	13
11 Information Management	5	1	3	9	6	2	2	10
12 Social and Profess. Issues	8	3	4	15	7	4	4	15
13 Software Engineering	11	10	6	27	8	11	4	23
14 Computational Science	1	2	7	10	0	1	4	5

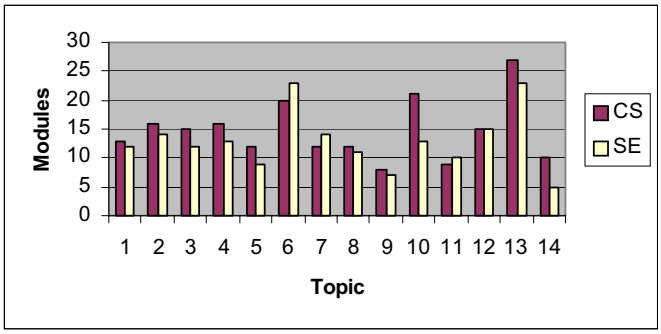

Figure 1: CS and SE Degrees Using the CS Curriculum Core Topics

[2] E.g. Computer Architectures I and Computer Architectures II would count as 2 modules.

Table 2: CS and SE Programs with regards to the ACM/IEEE Software Engineering Curriculum

Core Topics	CS Programmes				SE Programmes			
	Year 1	Year 2	Year 3	Total	Year 1	Year 2	Year 3	Total
1 Computing Essentials	13	2	0	15	12	0	0	12
2 Maths and Engineer. Fundamentals	6	0	0	6	7	1	0	8
3 Professional Practice	7	2	1	10	6	2	4	12
4 Software Modeling and Analysis	4	6	1	11	6	11	2	19
5 Software Design	10	6	2	18	6	10	2	18
6 Software Verification & Validation	1	2	2	5	0	3	2	5
7 Software Evolution	1	3	0	4	0	2	2	4
8 Software Process	5	3	2	10	6	3	3	12
9 Software Quality	1	3	1	5	1	3	2	6
10 Software Management	2	4	4	10	2	5	3	10
11 Systems and Applicat. Specialties	2	5	5	12	2	6	11	19

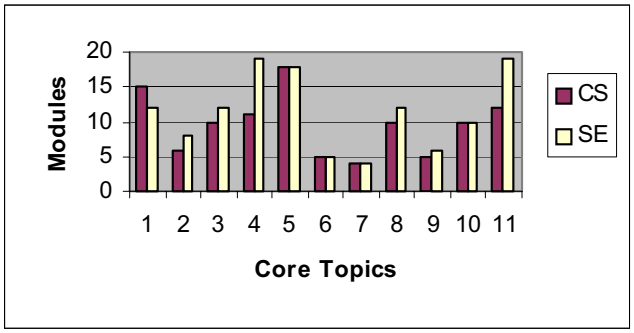

Figure 2: CS and SE Degrees Using the CS Curriculum Core Topics

4. Discussion and Conclusion

The results in section 3 reveal a number of interesting issues, some expected, and some less expected:

- As expected, there is greater emphasis on Software Modeling and Analysis in the SE programs than the CS programs.
- There is a strong presence on "intelligent systems" in CS degrees almost to the extent that this is a major characterizing difference between SE and CS programs in England.
- As expected, there is a strong presence of "Net-Centric Computing" for both types of programs as modules on web-based applications have become more popular
- There is relatively little material on mathematics, which is not surprising to those who are aware of the declining number of students studying mathematics at A level in the UK, but might be to those who are aware of the strength of the UK research community in theoretical computer science.

These are interesting points of discussion, but the most striking conclusion is that:

There is very little difference between the SE and CS programs currently offered in English Universities.

It could be argued that this situation exists because there was no clear model for SE programs until now. Although this might be a factor, the most likely reasons are historical and economical considerations. The number of applicants for SE programs in the UK is relatively low, with only 1051 students applying for SE compared to 13484 for CS programs in 2002 [15]. This in turn translates to only a few students on a SE program making it expensive to offer separate modules specifically designed to cover the most appropriate SE topics. Given the larger number of students on CS programs, it is much more economical to carry out a minimal variation of an existing CS program to produce a SE program. This is evident in the above results, where contrary to the recommendations of the ACM/IEEE SE curriculum, modules such as Operating Systems, Computer Architecture and Graphics are still strongly present in SE programs. Given the limited number of applicants for SE programs in England, the increasing demands to balance budgets and the relatively high staff-student ratios in Computing Departments in England, departments are less likely to adopt the ACM/IEEE SE model than the CS model. Suggestion to improve current content and delivery of SE programmes would include a compulsory industrial placement for SE students with a carefully selected project that will emphasis on those aspects that may not have been covered in the curriculum and a purely SE final year project to satisfy specified SE learning outcomes.

6. References

[1] ACM/IEEE-Curriculum 2001 Task Force, "Computing Curricula 2001, Computer Science ", December 2001, http://www.computer.org/education/cc2001/final/index.htm

[2] ACM/IEEE Computing Curricula 2003 Task Force, "Computing Curriculum-Software Engineering Public Draft 1", July 2003, http://www.computer.org/education/cc2001/final/index.htm

[3] P. Bourque and R. Dupuis (eds), "Guide to the Software Engineering Body of Knowledge", IEEE CS Press, Los Alamitos, 2001.

[4] A.J. Cowling, "What Should Graduating Software Engineers be Able to do?". Proceedings of the 16th Conference on Software Engineering and Education (CSEET'03), IEEE Computer Society, 2003, pp. 88-98.

[5] M. Daniels, X. Faulkner and I. Newman, "Open Ended Group Projects, Motivating students and preparing them for the 'Real words'", In proceedings of the 15th Conference on Software Engineering Education (CSEE&T), IEEE Press, 2002, pp. 128-139.

[6] S.L. Duggins and B. B. Thomas, "An Historical Investigations of Graduate Software Engineering Curriculum", Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET'02), IEEE Press 2002, pp.78-87.

[7] P. Freeman, A.I. Wasserman and R.E. Fairley, "Essential Elements of Software Engineering Education", In proceedings of the 2nd International Conference on Software Engineering, 1976, pp.116-122.

[8] P. Freeman, "Essential Elements of Software Engineering Education Revisited", IEEE Transactions in Software Engineering, SE-13, 1987, pp. 1143-1148.

[9] P. Naur and B. Randell, (Eds) "Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee", (7-11 October 1968), 1969.

[10] D.L. Parnas, "Software Engineering Programs are not Computer Science Programs", IEEE Software, November/December 1999, pp. 19-30.

[11] M. Shaw, "We Can Teach Software Better", Computing Research News, Vol. 4, Issue 4, Sep. 1992 pp.2-12.

[12] M. Shaw, "Software Engineering Education: A Road Map", In Proceedings of 22nd International Conference on Software Engineering (ICSE), ACM 2000.

[13] The Quality Assurance Agency for Higher Education, "The Framework for higher education qualifications in England, Wales and Northern Ireland", http://www.qaa.ac.uk/crntwork/nqf/ewni2001/contents.htm. January 2001

[14] M. Towhidnejad and T. B. Hilburn, "Software Quality Across the Curriculum", Proceedings of the 15th Conference on Software Engineering Education and Training , pp. 268-271, IEEE Press, 2002

[15] UCAS, www.ucas.ac.uk

[16] UCAS Subjects Data-Sets, 2002, http://www.ucas.ac.uk/figures/archive/subject/index.html

[17] C. Wohlin and B. Regnell, "Achieving Industrial Relevance in Software Engineering Education", Proceedings of the 12th Conference on Software Engineering Education and Training, pp. 16-25, IEEE Press 1999.