How does Software Engineering fit into a College of Engineering?

Thomas B. Hilburn, Massood Towhidnejad

Embry-Riddle Aeronautical University
hilburn@erau.edu, towhid@erau.edu

Abstract

Software Engineering (SE) is a new degree in most academic environments. Currently, there are less than 25 undergraduate SE degrees offered by US universities, and only six have been accredited by ABET. There are a number of challenges for faculty and departments who are offering these SE degrees. Some of these challenges include a) finding qualified faculty, b) designing an appropriate curriculum that serves the stakeholders needs, c) satisfying accreditation criteria, which are still evolving d) determining the difference between software engineering and computer science, e) and getting SE accepted as a "real" engineering discipline by more traditional engineering colleagues. It is clear that SE has a long way to go, to reach the state of stability and recognition that most traditional engineering degrees now enjoy. In this paper, we attempt to address the challenge of acceptance of software engineers as a "real engineers". We will also share with you some of our own experiences, when we were asked to move to the college of engineering and the problems we experienced and how we overcame them.

Introduction

It has been almost forty years since the first organized, formal discussion of software engineering (SE) as a discipline took place at the 1968 NATO Conference on Software Engineering [4]. Since that time there have been numerous studies and analysis of software development as an engineering profession [1, 5, 7, 9, 11]. Even though software engineering has made great progress and the term "software engineering" is now widely used in industry, it is still not yet well-defined and its professional content, practices, and certification criteria are not universally agreed upon. There is confusion and controversy over the relationship between computer science and SE, and some would even dispute that SE is engineering. A more generous attitude might be that the discipline is still new relative to more traditional engineering fields; and, thus, software engineering is simply not yet mature.

In spite of this immaturity (or maybe because of it), there is a large demand by software organizations for professionally trained software engineers. This is because of the ubiquitous nature of software – every part of human endeavor (business, education, entertainment, law, medicine, transportation, etc.) seems to be dependent on the effective and efficient development of software. Modern software systems are large (millions of lines of code) and complex (thousands of units that must interact with each other and with other entities). Lives and livelihoods depend on software; and unfortunately, there are serious problems in the cost, timeliness, and quality of many software products [10]. All of this points to the need to integrate and incorporate engineering practices into the development of software systems. That is, industry sees the need to move away from ad hoc, undisciplined programming techniques to the "engineering" of software systems. In the past decade the demand for graduates with software engineering knowledge and skill (as opposed to students with more conventional computer science degrees) has grown dramatically and the Bureau of Labor Statistics rates software engineering as one of the fastest growing professions in this decade [6]. This paper describes how undergraduate programs in software engineering are being developed and how they compare to more traditional academic engineering programs. In particular, we discuss how software engineering fits into a college of engineering.

The Nature of Software Engineering and Other Engineering Disciplines

One of the obstacles to the advancement of software engineering is an understanding of how it relates to more traditional engineering disciplines (such as chemical, civil, electrical and mechanical engineering). The 2001 ACM/IEE-CS SE Curriculum project, "SE 2004" [3] cited the following characteristics that were common to all engineering disciplines, including software engineering:

- Engineers proceed by making a series of decisions, carefully evaluating options, and choosing an approach at each decision-point that is appropriate for the current task in the current context. Appropriateness can be judged by tradeoff analysis, which balances costs against benefits.
- Engineers measure things, and when appropriate, work quantitatively; they calibrate and validate their measurements; and they use approximations based on experience and empirical data.
- Engineers emphasize the use of a disciplined process when creating a design and can operate effectively as part of a team in doing so.
- Engineers can have multiple roles: research, development, design, production, testing, construction, operations, management, and others such as sales, consulting, and teaching.
- Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is key to engineering.
- Engineers, via their professional societies, advance by the development and validation of principles, standards, and best practices.
- Engineers reuse designs and design artifacts.

However, there are two key differences between products designed by software engineers (software products) and other engineers (bridges, automobiles, electrical components, etc.). Software is abstract rather than physical and it deals with discrete rather than continuous entities. The intangible nature of software affects its visibility, its changeability, and its lack of conformance to the laws of nature. When designing physical products an engineer can express the design of the product in terms of visual and physical models that are meaningful to clients, other engineers, and those responsible for constructing or manufacturing the product. Making software “visible” is a major challenge for software engineers. A number of software modeling methods and techniques have been developed and are widely used to produced higher quality software; however, even though such models produce multiple “views” of the software and help us understand how they will be constructed, the models are abstract in form and nature, and are not directly related to the physical world.

One of the advantages and curses of software development is the ease with which software can be changed. For example, extra features can be added to a software application by simply changing the source code for the product. While changing the design of a physical product (e.g., increasing the maximum load supported by a bridge or adding a fifth wheel to an automobile) is a major re-design and construction activity. Unfortunately, software is often changed without updating the design documentation or thoroughly testing the new product produced, which leads to problems in quality and future maintenance costs.

The fact that software deals in discrete, finite data both simplifies and constrains software in a way that distinguishes it from more established fields of engineering. These fundamental differences influence the type of preparation needed for engineering practice and professional competence. For example, mechanical engineers need a firm grounding in continuous mathematics and the physical sciences, while software engineers need a solid foundation in discrete mathematics and computer science.

Initial Professional Preparation

As with other engineering disciplines, the initial professional preparation of software engineers must begin with undergraduate programs that provide a solid foundation in mathematics, basic science, engineering science and design. The general criteria specified by the Engineering Accreditation Commission of ABET [2] provide commonality between undergraduate engineering programs. The quantitative requirements for math, science, and engineering are the same; and the importance of a major design experience is stressed for all engineering programs. These criteria have the effect of producing curricula that address common needs for all engineering programs and provide students:

- a strong foundation in math and science
- significant experience in problem solving
- the basis for establishing an engineering mindset and a commitment to the profession

However, if one looks at the specific program criteria for SE in comparison to other more traditional programs there are some clear differences. Most of the program criteria specifically require or strongly imply the need for course work in continuous mathematics, through differential equations. They also refer to specific natural sciences (physics, chemistry or biology) and closely related domain knowledge (“aerospace materials”, “electronic devices”, “solid and fluid components”, etc.). On the other hand the SE program criteria make no reference to continuous mathematics or to natural science, but specify knowledge about discrete mathematics. Most significantly one notices the SE program criteria do not have specific reference to a domain of application. The capabilities referred to (“the ability to analyze, design, verify, validate, implement, apply, and maintain software systems”) are generic and could be applied to any application domain where software is needed. There are two reasons for this. First, as stated before, software is used everywhere in today's world; and second, and more critical to curriculum design, is the abstract nature of software. Abstraction and information hiding are the key principles behind effective software design and permeate all phases of the software development life-cycle. These factors influence the design and implementation of SE programs that have a different "look and feel" than conventional engineering programs.

Software Engineering in the College of Engineering

One might wonder which college should house the software engineering (SE) program. There are number of good answers to this question, and it all depends on the nature of the university and the college, which offers the program. One could argue that the SE program should be housed as part of the College of Art and Science (CoAS). Supporters of this argument use the fact that the SE program is an offspring of a computer science program, therefore it is natural to have SE as part of the (CoAS). On the other hand, there is group that argues that SE should be part of the College of Engineering (CoE); the main argument this group uses is the fact that engineering is used as part of the degree name. Finally, there is a group which argues that software engineering, computer science, information technology, information systems, and other computing related degrees should all be grouped together and offered by a college which is specialized in computing, for example a college of computer and information science. Of course, each of these proposals has its own advantages and disadvantages. In this section we discuss some of the advantages and disadvantages of being part of the college of Engineering.

There are number of advantages in having software engineering as part of the college of engineering. Almost all the modern engineering applications interact with computers in one form or another. Almost all engineers use software applications as part of their day-to-day job performance. For example, civil, mechanical, electrical and aerospace engineers use software simulation to evaluate the performance of the systems they design. In addition, there are a number of engineering applications that have computers as an integral component of the overall system under development, and this requires engineers to have a more in-depth understanding of the computer software and hardware. For example, mechanical engineers now commonly use embedded computers to control their systems (e.g., in automobiles) and aerospace engineers design aircraft, incorporating numerous embedded computers (e.g., 1280 in the Boeing 777). Finally, it is common practice for engineers to use computers to diagnose system problems. Hence, it seems obvious that engineers are required to either understand how they use computers (hardware and software) as part of their day-to-day operation, or be able to understand and interact with the people who do. This is also true for the software engineers who must work with other engineers, and therefore they need to understand their application domain. Therefore, having software engineering as part of the college of engineering, will flourish this culture from the very beginning. This culture can thrive through curriculum development and in other settings (e.g., research, colloquiums, and student activities). One idea would be the involvement of SE students and students form other engineering disciplines in an inter-disciplinary senior design project. Students have the opportunity to learn about each other's domain, constraints and complexities – invaluable lessons for the workplace. In the same vein, an undergraduate research project can provide the opportunity for students from different engineering programs, including SE students, to work on a common project.

On the other hand, there are number of disadvantages for having software engineering as part of the college of engineering. The most critical disadvantage is the lack of understanding and acceptance of the software engineering as an engineering discipline. There is a resistance on the part of many traditional engineers to accept software engineering as a “real” engineering discipline. Another disadvantage is the lack of understanding of what is needed for a software engineering curriculum. This is a special problem in colleges that require a common core of courses for all engineering students in the first year or two. This can result in software engineering being viewed as different from the rest; hence, the question of “Is this really engineering?” arises. Also, in order for SE to fit into a common core it may have to sacrifice curriculum quality. For example, other engineering program rarely will require a programming course in their first semester, or even worst require two programming classes during their first year, but it is critical for software engineering students to begin programming as soon as possible.

Another problem with software engineering being part of the college of engineering is the uniqueness of the SE laboratory. The SE program requires computers and appropriate software in their laboratory, and they occasionally require a special laboratory set up (i.e., area where students can work on a team project with multiple computers and work space for each team.) Also, there are situations, where the SE program requires classes with a computer for each student, in order to gain hands-on experience, and these requirements are often foreign to traditional engineering programs, therefore acquiring funding can be more difficult. Finally, hiring faculty can be much more difficult. Traditional engineering degrees require faculty with a terminal degree in the specific engineering field, and in some cases, they also require PE (Professional Engineering); this is not true or possible in the case of the software engineering. There are only two PhD program in SE in the U.S. and the PE for software engineering only exists in the state of Texas.

It is obvious that there are some advantages and disadvantages for being part of an engineering college, and at this point, it may seem like that the disadvantages outweigh the advantages; however, with appropriate awareness and dialogue, we are sure we can get to the point that there will be far more advantages than disadvantages.

Sharing our experience, the good, the bad, and the ugly

Finally, we would like to share some of our experiences over that last three years in becoming part of a “traditional” engineering college. Before we do so, let us discuss the nature of our university. Our university is an aeronautical university where the two biggest programs are Aerospace Engineering, in the College of Engineering, and Aeronautical Science, in the College of Aviation. Given the main concentration of our university in aviation and aerospace, some faculty have questioned whether we need a degree in software engineering. Although this was a minority view, it took some convincing on our part to show the relationship between software engineering and aviation and aerospace. Fortunately, the majority of our upper administration recognized this relationship very quickly, and they become the supporters of the program.

Right now, our biggest problem is with recruiting students to the SE program. There are three main reasons for our enrollment problem. One relates to the national trend, where the enrollment in software engineering and computer science has declined in recent years (probably strongly related to the dot com bust), but fortunately it seems like, this factor is abating. The second reason is that, as discussed in earlier sections, software engineering is a new discipline and still not widely understood (e.g., SE is not listed in most college guides). This lack of understanding makes the selection of SE by high school students problematic. Finally, the aviation/aerospace nature of our university may dissuade a student interested in SE. Although we have strategies for addressing the enrollment problem, in the short term, a small SE student population results in multiple preparations for faculty and a higher workload. This is even worst in the case of software engineering, where we are dealing with a rapidly changing content and the lack of understanding by our colleagues for need of periodically refreshing material in our classes. This is a constant dilemma/battle that we have to deal with in our college.

The process of fitting the software engineering program to the common curriculum was also an exciting period, where for a while we were branded as “not a team player”. Fortunately, we were able to work closely with the college curriculum committee to come up with a compromise that was acceptable to all.

A minor, but frustrating problem was dealing with a pocket of resistance by a few “traditional” engineering faculty, where they stated that SE is not a legitimate engineering discipline, and compared software engineering to “sanitation engineering” and “underwater basket weaving engineering”. Fortunately, some of them are no longer with the college (retirement), and the remaining ones have turned into a silent minority.

Conclusion

Although software engineering faces numerous challenges, both as a profession and as a degree program, the importance of the discipline to the future of society and to other engineering activities insures not only its survival but also its growth and advancement. We have pointed to some of the pros and cons of the placement an SE program in an engineering college and shared some of our own experiences. We believe the need for good engineering practice as part of software development is paramount and the association with other engineering programs will only aid and abet this goal.

References

[1] Abran, A., et al, *Guide to the software engineering body of knowledge: 2004 version*, IEEE Computer Society, 2004; available at http://www.swebok.org/.

[2] Accreditation Board for Engineering and Technology, *Criteria For Accrediting Engineering Programs* (Effective for Evaluations During the 2004-2005 Accreditation Cycle), November 2003.

[3] ACM/IEEE-Curriculum 2001 Task Force, Computing Curricula, Software *Engineering 2004*, June 2004. (http://sites.computer.org/ccse/)

[4] Bauer, F.L., "Software Engineering", *Information Processing*, 71, 1972.

[5] Brooks, Fred P., *The Mythical Man-Month*, 20th Anniversary Edition, Addison Wesley, 1995.

[6] Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook*, 2004-05 Edition, http://bls.gov/oco/.

[7] Ford, G. and Gibbs, N. E., A *Mature Profession of Software Engineering*, CMU/SEI-96-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[8] Hill, Susan and Johnson, Jean, *Science and Engineering Degrees: 1966-2001*, National Science Foundation, 2004.

[9] Shaw, Mary, *Prospects for an Engineering Discipline of Software*, IEEE Software, vol 7, no 6, November 1990, pp.15-24.

[10] Standish Group, *The Standish Group Report: Chaos*, 1995. (http://www.scs.carleton.ca/~beau/PM/Standish-Report.html)

[11] Wasserman, Tony, Toward a Discipline of Software Engineering, *IEEE Software*, 13:6:23-31, Nov 1996.

Biography

THOMAS B. HILBURN is a professor in the Department of Computer and Software Engineering at Embry-Riddle Aeronautical University. His current interests include software engineering, formal methods, individual and team software processes, and computer science and software engineering education. Since 1996 he has served as a visiting scientist at the Software Engineering Institute, Carnegie Mellon University.

MASSOOD TOWHIDNEJAD is a professor and chair of Department of Computer and Software Engineering at Embry-Riddle Aeronautical University. His current interests include software engineering, software quality assurance and testing, and software engineering education.