Recommended Skills and Knowledge for Software Engineers

Steve Tockey

ABSTRACT

One goal of this paper is to put forward a proposition that computer science and software engineering are distinct but related subjects, and to clearly define how they are related. Another goal is to offer a recommendation on a set of skills and knowledge that could serve to distinguish proficient, "industrial-strength" software engineers. One potential use of this set of recommended skills and knowledge is as input to the development of a standardized software engineering curriculum that would produce graduates who would be highly valued by the software industry.

1. INTRODUCTION

There is reasonable agreement across the industry on what constitutes an appropriate set of skills and knowledge for computer science. For example, the Computing Sciences Accreditation Board has published its "Criteria for Accrediting Programs in Computer Science" [1]. A survey of curricula available on the World Wide Web [2], however, shows that there is little agreement on what constitutes appropriate skills and knowledge for software engineering. Coupled with this is the fact that there is little agreement on the relationship between computer science and software engineering. The net result is that software-engineering-degreed graduates, even those with the same degree title but from different institutions, often have widely varying skills and knowledge. This makes it difficult for prospective employers to assess the real value of a software engineering degree.

2. COMPUTER SCIENCE VERSUS SOFTWARE ENGINEERING, FROM FIRST PRINCIPLES

This section proposes a clearly defined relationship between computer science and software engineering. This proposal is based on first principles in that it calls on widely accepted definitions of the basic vocabulary, that is, the terms "science" and "engineering."

Science is defined as:

> . . . a department of systematized knowledge as an object of study; knowledge or a system of knowledge covering general truths or the operation of general laws, especially as obtained and tested through scientific method. [3]

The Accreditation Board of Engineering and Technology (ABET) is the recognized authority for accrediting engineering and technology degree programs at colleges and universities in the United States. ABET defines engineering as:

> . . . the profession in which a knowledge of the mathematical and natural sciences gained by study, experience, and practice is applied with judgement to develop ways to utilize, economically, the materials and forces of nature for the benefit of mankind. [4]

Comparing and contrasting these definitions shows that science is the pursuit of knowledge and engineering is the application of that knowledge for the benefit of people. For example, chemistry as a science is concerned with expanding our knowledge of chemical processes in order to better understand and explain phenomena that can be observed in the universe. Chemical engineering, on the other hand, applies the knowledge derived from this "chemical science" to filling human needs. Basic to chemical engineering is an understanding of the body of chemical theory. But, in addition, chemical engineering calls upon the practical aspects of chemical processes, such as the design of pressure vessels and waste-heat-removal mechanisms, together with an understanding of (engineering) economy.

Thus, the science branch and the engineering branch of a technical discipline are related but distinct. The science branch is concerned with the continued expansion of the body of theoretical knowledge about that discipline, whereas the engineering

branch is concerned with the practical and economical application of that same theoretical knowledge. The following equation is proposed to be a (possibly over-) simplified description of the general relationship between science and engineering:

$$\text{Engineering} = \text{Scientific theory} + \text{Practice} + \text{(Engineering) Economy} \quad (1)$$

Based on the definition of science, above, computer science can be defined as "a department of systematized knowledge about computing as an object of study; a system of knowledge covering general truths or the operation of general laws of computing especially as obtained and tested through scientific method."

Based on the ABET definition of engineering, software engineering can be defined as ." . . the profession in which a knowledge of the mathematical and computing sciences gained by study, experience, and practice is applied with judgement to develop ways to utilize, economically, computing systems for the benefit of mankind."

Thus, from Equation 1 we can derive

$$\text{Software Engineering} = \text{Computing theory} + \text{Practice} + \text{(Engineering) Economy} \quad (2)$$

Both computer science and software engineering deal with computers, computing, and software. The science of computing, as a body of knowledge, is at the core of both. Computing science is concerned with computers, computing, and software as a system of knowledge, together with the expansion of that knowledge. Software engineering, on the other hand, should be concerned with the application of computers, computing, and software to practical purposes, specifically the design, construction, and operation of efficient and economical computing systems.

3. RECOMMENDED SOFTWARE ENGINEERING SKILLS AND KNOWLEDGE

Based on the Software Engineering Institute's Capability Maturity Model [5] and their recommended graduate software engineering curriculum [6], combined with the author's 20+ years of combined industrial and academic software experience, the following is offered as a recommended set of skills and knowledge for software engineers. These skills and knowledge are believed to enable the proficient design, construction, and maintenance of cost-effective computing systems. These skills and knowledge are also believed to characterize "proper professional practice" for software engineers; that is, nonawareness is believed to be correlated with either a decrease in an individual's proficiency or a decrease in the cost-effectiveness of the resulting software. All other things being equal, a person who possesses such skills and knowledge should be considered more valuable to a software organization than a person who does not.

It should be noted that with the exception of Engineering Economy, the skills and knowledge recommended here are largely consistent with those presented in Bourque et al. [7], Hilburn et al. [8], Lethbridge [9], Cowling [10], and NWCET [11]. The differences are largely just the grouping of skill and knowledge kernels into broader "knowledge areas."

The dictionary [3] provides appropriate definitions for the intended use of the terms "skill" and "knowledge" in this paper:

> Skill—a learned power of doing something competently; a developed aptitude or ability.
>
> Knowledge—facts or ideas acquired by study, investigation, observation, or experience.

To be sure, this recommendation describes a vision of an "ideal" software engineer. No single person should be expected to live up to this ideal (i.e., be at an expert level in doing and knowing everything identified below). Individuals should be expected to have at least broad, but possibly shallow, proficiency across many of the skill and knowledge kernels and much more detailed proficiency in one or more specific areas of interest to them. Rather than saying that *every* software engineer must be proficient in all of the skills or knowledge items listed here, the intent is that *at least one* software engineer on each software project ought to be proficient in each relevant skill or knowledge kernel. Using a professional American football team as an analogy, not every team member is a proficient quarterback. Not every team member is a proficient defensive back. And not every team member is knowledgeable about the opposing team's strategies. But at least one team member is probably a proficient quarterback. At least one other team member is probably a proficient defensive back. And still one other team member is probably knowledgeable about the opposing team's strategies. The team, as a whole, leverages off the proficiencies of the individual team members.

The presentation of the skills and knowledge is organized according to Equation 2. Some packaging of these skills and knowledge into undergraduate and/or graduate curricula would, of course, be required. However, we wish to leave this packaging to the colleges and universities themselves because they are the most qualified to do so.

3.1. Computing Theory

Dr. Richard Feynman, one of the world's top physicists, was once asked how he was able to come up with so many good ideas. His response was that he came up with as many ideas as he could and then he threw away the bad ones. This seemingly off-hand comment underscores the importance of computing theory to software engineering. Knowledge of computing theory allows software engineers to:

- Propose a larger number of diverse designs than would otherwise be possible
- Identify and discard proposed designs that could not work (because they violate some known theory) earlier than otherwise possible

The remaining theoretically viable designs can then be evaluated on an economic basis to arrive at cost-effective designs.

Computer science and discrete mathematics provide the relevant theory of computing. Table 1 shows the recommended skills and knowledge for computing theory.

3.2. Software Practice

Software practice addresses the more practical, day-to-day issues encountered in industrial software settings. This subject area is broken down into several subareas as follows.

Table 2 shows the set of recommended skills and knowledge that might be classified under "software product engineering." It should be stated explicitly that these skills and knowledge apply as much to software maintenance as they do to software development.

Table 3 shows the set of recommended skills and knowledge that might be classified under "software quality assurance" (SQA). Note that contrary to what is found in many SQA courses, there is more to SQA than just software testing.

Table 1. Recommended computing theory skills and knowledge

Programming language concepts
Data structure concepts
Database system concepts
Relational algebra
Operating system concepts
Software architectures
Computer architectures
Automata theory and Petri nets
Computability theory and Turing machine theory
Complexity theory
Linguistics and parsing theory
Computer graphics
Set theory
Predicate logic
Formal proofs
Induction

Table 2. Recommended software product engineering skills and knowledge

Requirements, analysis, and equirements engineering
Software design
Code optimization and semantics preserving transformations
Human–computer interaction, and usability engineering
Specific programming languages
Debugging techniques
Software–software and software–hardware integration
Product family engineering techniques and reuse techniques
CASE/CASE tools

Table 3. Recommended software quality assurance skills and knowledge

Task kick-offs, previews, and readiness reviews
Peer reviews, inspections, and walk-throughs
Software project audits
Requirements tracing/Quality Function Deployment (QFD)
Software testing techniques
Proofs of correctness
Process definition and process improvement techniques
Statistical process control
Technology innovation

Just designing and constructing a software system may itself be insufficient. The software product needs to be packaged and delivered to the customer in a form that they can use. The customer may need to be prepared to receive it. The customer may also need help in using, maintaining, or repairing the product. Table 4 shows an often-forgotten aspect of software engineering: "software product deployment."

Software projects are often resource-critical. People, time, money, equipment, and so on. all need to be coordinated to obtain the maximum payoff from the corporation's investment. "Software engineering management" is intended to provide the needed coordination both within the software organization itself and between the software organization and its neighboring organizations. Table 5 shows these recommended skills and knowledge.

3.3. Engineering Economy

The dictionary defines "economy" as "thrifty and efficient use of resources" [3]. Engineering economy is applied microeconomics, where the fundamental question is, "Is it in the best interest of the enterprise to invest its limited resources in a proposed technical endeavor, or would the same investment produce a higher return elsewhere?" [12].

Table 4. Recommended software product deployment skills and knowledge

User documentation techniques
Product packaging techniques
System conversion techniques
Customer support techniques
General technology transfer issues

Table 5. Recommended software engineering management skills and knowledge

Risk assessment and risk management
Project planning
Alternative software lifecycles
Organizational structures
Organizational behavior
Project tracking and oversight
Cost management, schedule management, and resource management
Metrics, goal–question–metric paradigm, and measurement theory
Configuration management and change management
Supplier and subcontract management
Effective meeting skills
Effective communication skills
Negotiation skills

From a business standpoint, profit is not only an organization's goal; it is necessary for its survival. The ultimate aim of engineering is to create the most income from the least expense, thus maximizing profit.

From a government perspective, engineering economy is still very important. The purpose of the government is to serve its citizens, primarily in terms of national defense and general welfare. The government is funded through taxation. Increasing taxation slows the economy and subtracts from the general welfare. The aim in government should be to provide the maximum benefit to the most people while keeping taxation to a minimum. This same principle applies to *any* nonprofit organization: "How can we satisfy people's needs in the most cost-effective manner?"

Leon Levy makes some significant observations regarding engineering economy and its relevance to software engineering:

> [S]oftware economics has often been misconceived as the means of estimating the cost of programming projects. But economics is primarily a science of choice, and software economics should provide methods and models for analyzing the choices that software projects must make. [13]

And,

> In any software project there is always a balance between short term and long term concerns ... economic methods can help us make enlightened choices.

The relevance of engineering economy to software engineering is described in much more detail in [14], which also includes a discussion of a software engineering economy course offered in the Masters of Software Engineering curriculum at Seattle University.

Table 6 shows the skills and knowledge generally found in engineering economy.

3.4. Customer and Business Environment

In truth, Equations 1 and 2 are somewhat incomplete. Software engineers cannot offer products and services that will be cost-effective to customers without understanding both those customers and how the products and services impact the customer's business. Software engineers need the following knowledge:

- Who is the customer and what is their business?
- What do they use our products and services for?
- When, where, and why are our products and services used?
- Are our products and services being used in a way different than originally intended? If so, why?
- How do our products and services affect the customers' business?
- What external restrictions or regulations impact the ability to deliver products and services to the customer(s)?

We would not recommend, nor expect, that these knowledge kernels be obtained from colleges and universities. Such knowledge is very company-specific and should be left at that level. However, Table 7 shows a set of additional recommended skill and knowledge kernels under the "customer and business environment" subject area that could be delivered by a college or university.

Table 6. Recommended engineering economy skills and knowledge

Time value of money (interest)
Economic equivalence
Inflation
Depreciation
Income taxes
Decision making among alternatives
Decision making under risk and uncertainty
Evaluating replacement alternatives
Evaluating public activities
Breakeven
Optimization

Table 7. Recommended customer and business environment skills and knowledge

Customer satisfaction assessment techniques
Competitive benchmarking techniques
Technical communication
Intellectual property law
Ethics and professionalism

4. PRACTICAL IMPLICATIONS

This section presents some practical implications of the proposals in the previous sections. There is no doubt that there will always be a need for qualified computer scientists to continue the advancement of computing theory. To be sure, every contemporary recognized engineering discipline has a corresponding science component populated by dedicated researchers. The contemporary computer science curriculum appears to be largely adequate in meeting this need. But the software industry also has a distinct need for "a practitioner who will be able to rapidly assume a position of substantial responsibility in an organization" [6].

Providing such qualified practitioners should be the primary goal of software engineering degree programs. It is hoped that the skills and knowledge recommended above can form the basis of a standardized curriculum for software engineering degrees. We are looking to initiate (or join, if one already exists) some kind of joint industry–academic forum to turn this recommendation from simply a wish list into real curricula.

5. SUMMARY

This paper proposed that computer science and software engineering are different but related subjects and defined how they are related and how they are different. This paper also offered a recommendation on a set of skills and knowledge that could serve to distinguish proficient, "industrial-strength" software engineers. It is our hope that this recommended skill and knowledge set could serve as input in the development of a standardized software engineering curriculum that would produce graduates who would be highly valued by the software industry. Software engineering graduates with such skills and knowledge would be able to rapidly assume positions of substantial responsibility in software organizations and those organizations would be able to appreciate the real value of that software engineering degree.

ACKNOWLEDGMENTS

The author gratefully acknowledges Rockwell Collins, Inc., and, in particular, Roger Shultz for the support given to the work leading up to this paper.

REFERENCES

1. Computing Sciences Accreditation Board, "Criteria for Accrediting Programs in Computer Science in the United States," Computing Sciences Accreditation Board, Stamford, Connecticut, June, 1996 (http://www.csab.org/criteria96.html).
2. Dave Eichman and Gary Ford, "Software Engineering Programs," University of Houston-Clear Lake, Houston, Texas, May, 1996 (http://ricis.cl.uh.edu/virt-lib/se_programs.html).
3. *The Merriam-Webster Dictionary,* New Edition, Merriam-Webster, Springfield, Massachusets, 1994.
4. Accreditation Board of Engineering and Technology, "Criteria for Accrediting Programs in Engineering in the United States," Accreditation Board of Engineering and Technology, Baltimore, Maryland (http://www.abet.org/eac/98eac-criteria.htm).
5. Mark Paulk, Charles Weber, Suzanne Garcia, Mary Beth Chrissis, and Marilyn Bush, "Key Practices of the Capability Maturity Model (sm), Version 1.1," Technical Report CMU/SEI-93-TR-025, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, February, 1993 (http://www.sei.cmu.edu/pub/documents/93.reports/pdf/tr25.93.pdf).
6. Gary Ford, "1991 SEI Report on Graduate Software Engineering Education," Technical Report CMU/SEI-91-TR-2, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, April, 1991 (http://www.sei.cmu.edu/pub/documents/91.reports/pdf/tr02.91.pdf).

7. Pierre Bourque, Robert Dupuis, Alain Abran, James W Moore, Leonard Tripp, Karen Shyne, Bryan Pflug, Marcela Maya, and Guy Tremblay, "Guide to the Software Engineering Body of Knowledge: A Straw Man Version," University du Quebec a Montreal, Canada, September, 1998 (http://www.lrgl.uqam.ca/).
8. Thomas Hilburn, Donald Bagert, Susan Mengel, and Dale Oexmann, "Software Engineering Across Computing Curricula," (http://erau.db.erau.edu/~hilburn/sei-educ/guide-pub.htm).
9. Timothy C. Lethbridge, "A Survey of the Relevance of Computer Science and Software Engineering Education," in *Proceedings of the 11th Conference on Software Engineering Education and Training (CSEE&T '98),* IEEE Computer Society Press, 1998.
10. A. J. Cowling, "A Multi-Dimensional Model of the Software Engineering Curriculum," in *Proceedings of the 11th Conference on Software Engineering Education and Training (CSEE&T '98),* IEEE Computer Society Press, 1998.
11. Barbara Roll, Ed., *Building a Foundation for Tomorrow: Skill Standards for Information Technology,* NorthWest Center for Emerging Technologies, Bellevue Community College, 1997 (http://nwcet.bcc.ctc.edu).
12. G. Thuesen; W. Fabrycky, Engineering Economy, 8th ed., Prentice-Hall, Englewood Cliffs, NJ, 1993.
13. Leon Levy, *Taming the Tiger—Software Engineering and Software Economics,* Springer-Verlag, New York, 1987.
14. Steve Tockey, "A Missing Link in Software Engineering," *IEEE Software, 14,* 6, November/December, 1997.