Computer Science: Is It Really the Scientific Foundation for Software Engineering?

Stephen T. Frezza, *Gannon University*

There are significant differences between an engineering discipline founded on a natural science and one founded on a formal science.

Software engineering is often referred to as an "emerging" discipline—something that's relatively new and for which educational standards and definitions are needed. Fortunately, volunteers from the IEEE Computer Society and the ACM took on this task, with the result being the *Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering* (http://sites.computer.org/ccse/SE2004Volume.pdf). According to SE04:

> "Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems. . . . One particularly important aspect is that software engineering builds on computer science and mathematics. But, in the engineering tradition, it goes beyond this technical basis to draw upon a broader range of disciplines. . . . Software engineering . . . is different in character from other engineering disciplines, due to both the intangible nature of software and to the discrete nature of software operation. It seeks to integrate the principles of mathematics and computer science with the engineering practices developed to produce tangible, physical artifacts."

This definition, designed to support the development of undergraduate programs, emphasizes the importance of the relationship between software engineering and computer science. The "important aspect," that software engineering builds on computer science and mathematics, stands out as central to the definition of the discipline.

BASIC SCIENCE VS. ENGINEERING SCIENCE

From my own experience, I would say that software engineering builds on computer science and mathematics in the same way that my own degrees in electrical engineering build on physics and mathematics. Clearly, the professionals at the IEEE CS and ACM who drafted this language, reviewed it, and use it to develop undergraduate software engineering programs around the world have found this relationship useful. One particular point of utility was in helping fellow computing faculty understand the similarities and differences between computer science and software engineering. That is, until the practical question of engineering accreditation became a reality.

For all undergraduate engineering programs in the US, the organization responsible for an undergraduate engineering program's accreditation is ABET's Engineering Accreditation Commission. The EAC defines nine general criteria that must be met for the program to include the title "engineering" (www.abet.org/Linked%20Documents-UPDATE/Criteria%20and%20PP/E001%2009-10%20EAC%20Criteria%2012-01-08.pdf).

EAC Criterion 5 specifies the minimum expectations for an engineering program's required courses: "one year of a combination of college level mathematics and basic sciences

 Published by the IEEE Computer Society

(some with experimental experience) appropriate to the discipline." This is regularly interpreted to mean that 25 percent of the credits required of the program consist of mathematics and basic science courses. Historically, this has been interpreted as "natural science" and "mathematics" or other course offerings from traditional science and mathematics departments.

ABET clarifies this further: "The engineering sciences have their roots in mathematics and basic sciences but carry knowledge further toward creative application. These studies provide a bridge between mathematics and basic sciences on the one hand and engineering practice on the other."

While no official interpretation for a basic science is provided, the terminology is meant to distinguish a physics or geology course from engineering science courses such as thermodynamics or materials science. The rationale is probably to prevent undergraduate engineering programs from artificially declaring engineering courses as basic science and unbalancing the program with something less rigorous than is otherwise warranted.

The open question is where computer science fits in. Is it mathematics, a basic science, or an engineering science? Is it some mix of two or even all three? Here the more established engineering definitions, as evidenced by the EAC criterion, differ significantly from the software engineering definitions, as presented in SE04.

Reading the EAC criterion, the expectation might be that some, perhaps a significant, amount of computer science coursework would be considered as basic sciences appropriate to the discipline of software engineering—in particular, those computer science courses significantly grounded in mathematics, such as data structures, algorithms, computability, and formal methods. From this perspective, an undergraduate program would be expected to build its science and mathematics coursework solidly around computer science courses, particularly foundational ones.

Conversely, one could easily read the EAC criterion to view all computer science coursework as engineering science—after all, programming is a form of engineering, a creative endeavor that produces a real product. Computer science appears significantly disconnected from the scientific method that underpins the study of natural science. From this perspective, all computer science courses, even foundational ones, should be classified as "engineering science," not "mathematics and basic science."

One would like to think that this discrepancy is just an exercise in pointless academic epistemology. Unfortunately, it has significant ramifications for engineering education, particularly computing education. How computer science coursework is viewed by engineers and accrediting bodies fundamentally skews how undergraduate software engineering and other computing programs are built.

The reality is that all undergraduate engineering programs are under significant pressure to maximize the material and learning within the limited credit hours available in the program. Consequently, any undergraduate software engineering program built with computer science as its foundation instead of natural science and traditional mathematics would at best receive a "deficient" rating for EAC Criterion 5. Depending on your view of engineering as a discipline, this may or may not be a good thing.

If computer science is an engineering science that "bridges science and mathematics to engineering design," then it hardly qualifies as the scientific foundation for software engineering or any other form of engineering, at least from the ABET and similar perspectives. On the other hand, if it isn't just an engineering science, then its scientific content and methodology would need to be understood for well-formulated software engineering programs. Distinguishing computer science as basic science, engineering science, or some blend is critical for software engineering as both a discipline and an educational program.

Table 1. Some differences between science and engineering in the UK.

Science	Engineering
Goal: pursuit of knowledge and understanding for its own sake	Goal: creation of successful artifacts and systems to meet people's wants and needs
Key scientific process: discovery (mainly by controlled experimentation)	Corresponding engineering process: invention, design, production
Analysis, generalization, and synthesis of hypotheses	Analysis and synthesis of design
Reductionism, involving the isolation and definition of distinct concepts	Holism, involving the integration of many competing demands, theories, and ideas
Making more or less value-free statements	Activities always value-laden
The search for and theorizing about causes, such as gravity, electromagnetism	The search for and theorizing about processes, such as control, information, networking
Pursuit of accuracy in modeling	Pursuit of sufficient accuracy in modeling to achieve success
Experimental and logical skills	Design, construction, test, planning quality assurance, problem solving, interpersonal communication skills

COMPUTER SCIENCE IN ENGINEERING PROGRAMS

In the UK education system, engineering science has a more formal meaning and focuses on experi-

Table 2. Comparison of *Essential Cell Biology*'s table of contents with that of *Data Structures Using C++*.

Biology text		Engineering text	
1.	Introduction to Cells	1.	Software Engineering Principles and C++ Classes
2.	Chemical Components of Cells	2.	Object-Oriented Design (OOD) and C++
3.	Energy, Catalysis, and Biosynthesis	3.	Pointers and Array-Based Lists
4.	Protein Structure and Function	4.	Standard Template Library (STL) I
5.	DNA and Chromosomes	5.	Linked Lists
6.	DNA Replication, Repair, and Recombination	6.	Recursion
7.	From DNA to Protein: How Cells Read the Genome	7.	Stacks
8.	Control of Gene Expression	8.	Queues
9.	How Genes and Genomes Evolve	9.	Search Algorithms
10.	Manipulating Genes and Cells	10.	Sorting Algorithms
11.	Membrane Structure	11.	Binary Trees
12.	Membrane Transport	12.	Graph Algorithms
13.	How Cells Obtain Energy from Food	13.	Standard Template Library (STL) II
14.	Energy Generation in Mitochondria and Chloroplasts	A.	Reserved Words
15.	Intracellular Compartments and Transport	B.	Operator Precedence
16.	Cell Communication	C.	Character Sets
17.	Cytoskeleton	D.	Operator Overloading
18.	The Cell Division Cycle	E.	Header Files
19.	Genetics, Meiosis, and the Molecular Basis of Heredity	F.	Additional C++ Topics (Inheritance, Pointers, and Virtual Functions)
20.	Tissues and Cancer	G.	Problem Solving Using Object Oriented Methodology
—		H.	C++ for Java Programmers
—		I.	References for Further Study
—		J.	Answers to Odd-Numbered Exercises

mental design and analysis. Table 1 highlights the differences.

Taking some clues from the UK perspective, a working definition for engineering science realizable in the classroom would be a bridging or blend of science and engineering perspectives:

- discovery that leads to invention;
- hypotheses that support design;
- isolation of specific concepts to support the integration of multiple ones; and
- understanding sufficiency in model accuracy.

The practical question to answer would be where most computer science courses fall along this scale—science, engineering science, engineering. A cursory comparison of a natural science to a computer science textbook may help.

Table 2 shows a side-by-side comparison of the table of contents of *Essential Cell Biology* (B. Alberts et al., 3rd ed., Garland Science, 2009) with that of *Data Structures Using C++* (D.S. Malik, 2nd ed., Course Technology, 2009). I chose these based on their topics and their popularity on Amazon.com.

In looking at the 20 chapters of the biology text, most of these topics—at least the first 18—put a pedagogical focus on the transfer of knowledge of cell biology as it's currently understood, including the recognition of the structures involved and how these structures interact to form cell behavior. Most of the text focuses on terminology, definitions, and the many important processes and relationships that make up cell biology. Only in the last two chapters do the authors begin to shift the focus to the application of this general knowledge to practical issues, such as "Tissues and Cancer."

The table of contents for the data structures text is for a course that follows most introductory programming courses. It's what SE04 designates as part of computer fundamentals and is required of accredited computer science programs. In this text, an exactly opposite view is presented: the first two chapters focus on relating the material to engineering and design, and the following 11 chapters, as well as most of the appendices, focus on the transfer of knowledge of computing data structures and how these structures operate both mathematically as well as in a formal description language. Most of the text appears to focus on terminology, definitions, and the many impor-

tant processes and relationships that make up computing data structures.

Again, you could argue that what students do for homework in the data structures course is significantly different from that in cell biology. The questions and exercises in a computer science course invariably include the creation of artifacts, whereas the biology course does not. The goal is to teach students how to use data structures to build useful code; the artifacts bridge mathematics and science, so this coursework should be classified as engineering or, at the very least, engineering science.

This interpretation, unfortunately, stands at odds with the IEEE CS and ACM assertions in SE04's Guiding Principle 2 on computer science and its relationship to software engineering:

> "Software Engineering draws its foundations from a wide variety of disciplines. Undergraduate study of software engineering relies on many areas in computer science for its theoretical and conceptual foundations, but it also requires students to utilize concepts from a variety of other fields, such as mathematics, engineering, and project management, and one or more application domains. All software engineering students must learn to integrate theory and practice, to recognize the importance of abstraction and modeling, to be able to acquire special domain knowledge beyond the computing discipline for the purposes of supporting software development in specific domains of application, and to appreciate the value of good design."

From my own perspective as a software engineering educator, I suspect that the IEEE CS and ACM have a point. From the students' perspective, the data structures course is substantively much more like math or biology than engineering. The artifacts generated aren't about invention, design, or discovery, rather the work is primarily focused on understanding: What are data structures? How are they expressed? Which are more efficient than others in different situations? The application or bridging to engineering is a secondary, not a primary, focus of most coursework. From the student learning perspective, data structures aren't significantly different from biology—it's really the study of a formal science, not a predominantly engineering science.

Whether you agree with this or not, expect there to be significant differences between an engineering discipline founded on a natural science and one founded on a formal science. This certainly isn't surprising to most software engineers.

Stephen T. Frezza *is an associate professor of software engineering in the Computer and Information Science Department at Gannon University. Contact him at frezza001@gannon.edu.*

Editor: Ann E.K. Sobel, Department of Computer Science and Software Engineering, Miami University; sobelae@muohio.edu