"Any clod can have the facts, but having opinions is an art."
Charles McCabe, San Francisco Chronicle

Software engineering and computer science
Pei Hsia
University of Alabama in Huntsville

The first conference on software engineering was held at Garmisch, Germany in October 1968. That was the beginning of the emphasis on a new software discipline—software engineering. This term has been used for more than a decade now, but a great majority of people still seem to have problems in separating software engineering from computer science. This is due partly to the similarity of the topics appearing in "computer science" journals (e.g., CACM) on the one hand and "software engineering" journals (e.g., IEEE Transactions on Software Engineering) on the other. My purpose here is to stress the distinction between the two fields and point out their nonoverlapping nature. This may be beneficial to readers in their future considerations of software-related problems.

The term "computer science" is the result of an endeavor to transform computer programming from an art into a disciplined science.1 It grew gradually until it assumed its present-day stature, which includes at least the following branches or subfields: programming languages and systems, theory of computation, numerical analysis, artificial intelligence, and computer architecture. To the general public, however, computer science is simply a euphemism for computer programming.

On the other hand, software engineering is a direct product of programming experience. After many years of practice, we saw ever escalating software costs and continual failures of software projects. We began to realize that the way software was produced may not have been right. There was a need to look into new approaches so that we could improve software quality and reduce software cost. After looking into the achievements of computer hardware engineers, we believed that a similar engineering discipline was required for software. This was the beginning of software engineering. The term was coined with a specific goal in mind—improvement of software development and quality, but without a definition of a discipline to accomplish the goal. This was another reason why software engineering and computer science were so difficult to separate in the early 1970's.

Letting programmers "do their own things" in a group project always results in failure.

Other fields provide helpful analogies. Chemical engineering and chemistry are two disciplines illustrating the point. Many different chemical processes have been discovered in laboratories, but only a few of them can actually be used in chemical plants for production purposes. For example, there are many different processes which can extract gasoline from crude oil, but only a few of these are actually used and enlarged to a scale which can efficiently produce the required products. Not only must technical feasibility be demonstrated, but factors such as economic feasibility, reliability, and stability must be considered before scaling up a laboratory process into a production process.

The same relationship exists between software engineering and computer science. Not all programming techniques/algorithms can be used in large-scale programming environments/projects.

We have discovered that there are millions of different ways to produce a computer program to perform certain functions. However, letting programmers "do their own things" in a group project always results in failure. A colleague and I have found that a certain disciplined approach in programming can help beginning students produce better quality programs. The discipline used is a sequence of steps that the students are required to follow. Each step has well-defined start and end points. The discipline enables students to go through the program production process in a logical sequence of steps and realize how far they are from their goal and subgoals. The students are certainly not "doing their own things" nor are they exploring all the avenues of the programming art. The view of programming as a controllable production process is inherent in such a disciplined approach—what results is cost-effective code written with an eye toward the larger project. In computer science no one teaches programming in this light, because as long as students can write programs to communicate with the computer there is no problem. The emphasis is on exploration and experimentation. In contrast, software engineering is more concerned with the scaling up of the knowledge and techniques developed in computer science than with the knowledge itself. A compiler-compiler is undoubtedly an interesting idea in the computer science domain, but it has never been fully developed and used in any production project.

Electrical engineering and physics provide another comparison. Electronics is included in both electrical engineering and physics curricula, but the emphasis differs. Physics focuses more on the "why"—the principles and fundamental nature of electronics. Electrical engineering, however, accepts the phenomenon itself and emphasizes the application of the principles. Examples of applications, not explorations of theory, eventually enable electrical engineering students to be creative and to adapt the natural phenomena to new applications.
Similarly, in computer science a course in data structures involves the study of different data arrangements and concentrates on the theoretical aspect of their performance under certain algorithms. In a software engineering course, however, the study of data structures would emphasize their real-world applications. Both software engineering and computer science are based on a computer-centered world, just as both electrical engineering and electronics (physics) are based on an electrically-charged world, but computer science’s basic concern is the knowledge of computer function and operation while software engineering is the integration of proper techniques and knowledge leading to the solution of real-world problems. When one is developing a searching algorithm, he is in the domain of computer science, but when one is looking for a searching algorithm to be integrated into an information retrieval system, he is in the realm of software engineering.

The computer scientist’s responsibility is to develop computer knowledge and techniques, while the software engineer’s responsibility is to apply the techniques to produce specific software in a cost-effective manner. Computer scientists work with few requirements or constraints. Software engineers must solve problems involving conflicting requirements—certain levels of efficiency, reliability, complexity, cost, and performance may be required and may conflict with one another. Software engineers must search for the optimum solution. Software engineering, then, is centered around trade-off analysis and decision making on multiple requirements, while computer science is centered around technology development and theoretical studies.

To conclude these observations, I will paraphrase a well-known reference work’s distinction between science and engineering: The function of the computer scientist is to know, while that of the software engineer is to do. The computer scientist adds to the store of verified, systematized knowledge of the computer-centered world; the software engineer brings this knowledge to bear on practical problems.

The following ten figures can be obtained by removing extra line segments:


Was Edouard Lucas the first pattern recognizer?

Selim G. Akl
Queen’s University,
Kingston, Ontario

In the past 25 years the field of pattern recognition has attracted workers from a variety of disciplines, such as mathematics, engineering, science, and medicine. No one, however, seems to know who the first pattern recognizer was. Perhaps the nineteenth-century French mathematician, Edouard Lucas, author of the first published work in the field, was the real father of pattern recognition. In one of his famous books on recreational mathematics, a section entitled “The Shape of Numbers” deals with an aspect of what we would call today “character recognition.”

Lucas begins by observing that children should be trained early to form the numerals properly. He recommends that they use the simplest, clearest, and fastest way, “as they will have to write millions of them,” and briefly explains how to teach them.

The second part of the one-page essay discusses the origin of the shapes of the numerals we use. Lucas tells us that an old legend attributes these shapes to the ten figures which can be formed from a figure engraved on the stone of King Solomon’s ring. Indeed, from the pattern

It is interesting to see the similarity between these figures and the way numbers are shaped on today’s electronic displays.


*The term “pattern recognizer” was coined in 1976 by Professor L. Kanal while addressing the Third International Joint Conference on Pattern Recognition held in Coronado, California. It designates a person whose work is related in some way to the process of automatically recognizing patterns.