CHAPTER SEVENTEEN

Engineering & a Profession

Engineering can provide a life of genuine satisfaction in many ways, especially through ministering in a practical manner to the needs and welfare of mankind.

VANNEVAR BUSH

Engineers are saddled with much the same stereotype as computer programmers. They are regarded as boring and dull, and yet these boring and dull engineers are responsible for some of the most exciting developments in the world today. Putting a man on the moon, viewing the outer reaches of space from the Hubble Space Telescope, flying on modern jet aircraft, driving coast to coast in cars that operate nearly flawlessly, connecting to Internet sites throughout the world, enjoying theater-quality video presentations at home—these technological miracles are all predominately engineering accomplishments, the practical application of scientific principles.

Need for Engineering

Historically, professional engineering has been established in response to threats to public safety. Although we take the safety of modern bridges for granted, in the 1860s American bridges were failing at the rate of 25 or more per year.[1] Bridge failures and the loss of life they caused precipitated

creation of a stricter engineering approach to bridge design and construction. In Canada, engineering folklore holds that the collapse of the Quebec City bridge in 1907 catalyzed establishment of higher standards in all branches of Canadian engineering, which is symbolized today in the iron ring ceremony.[2] (I'll describe that ceremony in more detail in Chapter 19.) Engineers in Texas were licensed only after a boiler explosion in an elementary school killed more than 300 children in 1937.[3] The part that caused the explosion in 1937 has been replaced today by software.

Engineering differs from other professions in that doctors, dentists, public accountants, and lawyers generally provide their services to specific individuals or, in some cases, to specific corporations. Engineers tend to design *things* rather than provide services to individuals. Their responsibility is more often to society than to specific people. In this sense, software developers are more like engineers than they are like other kinds of professionals.

Software hasn't yet had its Quebec City bridge or its Texas elementary school boiler. But the potential is real. As any reader of the Forum on Risks to the Public in the Use of Computers and Related Systems[4] knows, software has already been responsible for many multi-million dollar losses, ranging from the ridiculous to the deadly. Tsutomu Shimomura parked on February 29, 1992, at a San Diego airport parking lot. When he returned six days later, his parking bill was $3,771. The parking software didn't recognize February 29 as a valid date.[5] In January 1990, approximately five million telephone calls were blocked over a nine-hour period because of a software error. The first space shuttle launch was delayed for two days because of a subtle programming error. The Mariner I space probe to Venus was lost because of an error in transcribing a guidance equation into software. In London, a computer dispatch system for ambulances was placed into operation before it was ready, collapsed completely, and caused delays as long as 11 hours. As many as 20 deaths were attributed to the new ambulance dispatch system. Iran Air Flight 655 was shot down by the USS *Vincennes*'s Aegis system in 1988, killing 290 people. The error was initially attributed to operator error, but later some experts attributed the incident to the poor design of Aegis's user interface.

Engineering and Art

Engineering's use of mathematics and science exposes it to the criticism that it is dry—that it saps the artistic elements out of structures that are engineered. The same criticism has been applied to software engineering. How true is this criticism? Does engineering exclude aesthetics?

Far from being antithetical to aesthetics, engineering is largely concerned with all aspects of design, including aesthetic aspects. Its designs aren't just limited to shapes and colors. Engineers design everything from electronic circuits to load-bearing beams to vehicles that land on the moon. As Samuel C. Florman says in *The Existential Pleasures of Engineering,* "Creative design is the central mission of the professional engineer."

Consider a comparison of two well-known buildings, the Reims Cathedral and the Sydney Opera House. The Reims Cathedral, shown in Figure 17-1, was completed about 1290; the Sydney Opera House, shown in Figure 17-2, in 1973. The Reims Cathedral was designed to use materials whose properties were understood (more or less) at the time.

The Sydney Opera House was constructed 700 years after the Reims Cathedral. As you can see in Figure 17-2, it's stylistically quite different from the Reims Cathedral. Its architects used modern materials such as steel and reinforced concrete, and they employed engineering techniques including computer modeling to determine how little material could safely be used.

Which building you prefer is a matter of taste, but which building can actually be built is a matter of engineering. It would be possible for modern builders to construct another Reims Cathedral, but it would not have been possible for 13th century builders to construct a Sydney Opera House. The reason the Sydney Opera House could not be built in the 13th century was not a lack of art, but the lack of engineering. We've all seen ugly buildings in which artistic considerations lost a battle with engineering economy, or in which aesthetics appear not to have been considered at all. Engineering without art can be ugly, but art without engineering may be impossible. Engineering does not constrain artistic possibilities. The *lack* of engineering constrains artistic possibilities.

FIGURE 17-1

Reims Cathedral, Reims, France. An example of art without very well developed engineering.[6]

FIGURE 17-2

Sydney Opera House, Sydney, Australia. An example of the dependence of art upon engineering.[7]

So it is with modern software systems. The level of engineering prowess determines how large a software system can be built successfully, how easy it will be to use, how fast it will operate, how many errors it will contain, and how well it will cooperate with other systems. Software includes many aesthetic elements, and software developers have no lack of artistic ambition. What we in the software industry sometimes lack is the engineering techniques that enable us to realize some of our grandest aesthetic aspirations.

Maturation of Engineering Disciplines

Disasters in older engineering fields have precipitated the professionalization of engineering practices in those fields. Of course, full-fledged engineering disciplines can't simply be willed into existence overnight. Mary Shaw at Carnegie Mellon University has identified a progression that fields go through before they reach the level of professional engineering. Figure 17-3 summarizes this maturation.

In the *craft* stage, good work is performed by talented amateurs. Craftsmen use intuition and brute force to create their widgets, whether their widgets are bridges, electric equipment, or computer programs. Some of their work is intended for sale to the public, but most is created solely for their own use. They have little or no concept of large-scale production

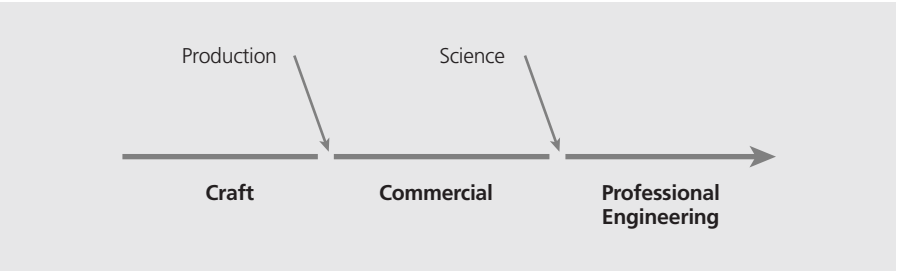

FIGURE 17-3
The progression of a discipline from craft to professional engineering.
Source: Adapted from "Prospects for an Engineering Discipline of Software"[8]

for external sale. Craftsmen tend to make extravagant use of available materials. The field progresses haphazardly; there's no systematic way to educate or train other craftsmen in the use of the most effective techniques.

Civil engineering (aqueduct and bridge construction) in first century Rome was a discipline in its craft stage, as was early computing in the 1950s and 1960s. Many software projects today still make extravagant use of available materials (staff time) and operate at the craft level.

At some point, the demand for the widgets increases beyond what isolated craftsmen can provide, and demand for greater production begins to influence the discipline. As the folklore becomes better understood, it's codified into written heuristics and procedural rules.

In the *commercial* stage, workers more carefully define the resources needed to support production. The stage is marked by a stronger economic orientation, and cost of goods may become an issue. Practitioners are trained to ensure consistent quality of the widgets they produce. Production procedures are systematically refined by changing different parameters to see what works and what doesn't.

The Reims Cathedral was built at a time when civil engineering was in its commercial stage. In software, many commercial-stage organizations achieve respectable levels of quality and productivity by making use of carefully selected, well-trained personnel. They rely on familiar practices and change them incrementally in pursuit of better products and better project performance.

Some of the problems encountered by commercial production can't be solved via trial and error, and, if the economic stakes are high enough, a corresponding science will develop. As the science matures, it develops theories that contribute to commercial practice, and this is the point at which the field reaches the *professional engineering* stage. At this point, progress arises from application of scientific principles as well as from practical experimentation. The practitioners working in the field at that point must be well-educated in both the theory and practice of their profession.

A Science for Software Development

Software science has been lagging behind commercial software development for years. Extremely large software systems were developed in the 1950s and 1960s, including the Sage missile defense system, the Sabre airline-reservation system, and IBM's OS/360 operating system. Commercial development of these large systems proceeded much faster than supporting research did, but practical applications advancing faster than science has been common in engineering. The airfoil wing section that allows airplanes to fly was invented just after it had been "proved" that no machine heavier than air could fly.[9] The development of thermodynamics followed the invention of the steam engine. When John Roebling designed the Brooklyn Bridge in the 1860s, the strength of steel cables was not well understood, and so he designed different parts of the bridge with safety margins as high as 6 to 1. This safety margin was an engineering judgment made in lieu of better theoretical knowledge.

The science that supports software development isn't as well defined as the physics that supports civil engineering. In fact it isn't even considered "natural science." It is what Herbert Simon calls a "science of the artificial"[10]—the knowledge areas of computer science, mathematics, psychology, sociology, and management science. A few software organizations regularly apply theories from these areas to their projects, but we are a long way from seeing universal application of these sciences of the artificial to software projects.

But are we really asking software science to provide the right things? For many classes of applications—inventory management systems, payroll programs, general ledger software, operating system design, database management software, language compilers (the list is nearly endless)—the same basic applications have been written so many times that these systems shouldn't require as much unique design effort as they seem to need. Mary Shaw points out that in mature engineering fields routine design involves solving familiar problems and reusing large portions of prior solutions. Often these "solutions" are codified in the form of equations, analytical models, or prebuilt components. Unique design challenges do present themselves from time to time, but the bread and butter of

engineering is the application of routine design practices to familiar problems.

The software world is still in the process of capturing many of its "solutions" in ways that are useful to the average practitioner. Many software project artifacts are potentially reusable, and many of them promise more potential to improve quality and productivity than the most commonly reused artifact, source code, does. Here is a short list of some project artifacts that can be reused:[11]

- Architectures themselves and software design procedures
- Design patterns
- Requirements themselves and requirements engineering procedures
- User interface elements and user interface design procedures
- Estimates themselves and estimation procedures
- Planning data, project plans, and planning procedures
- Test plans, test cases, test data, and test procedures
- Technical review procedures
- Source code, construction procedures, and integration procedures
- Software configuration management procedures
- Post-project reports and project-review procedures
- Organizational structures, team structures, and management procedures

At present, few of these project artifacts have been packaged into a form that the average organization can readily apply.

Science has not yet provided software development with a set of equations that describe how to run a project successfully, or that describe how to produce successful software products. Perhaps it never will. But science doesn't necessarily have to consist of formulas and mathematics. In *The Structure of Scientific Revolutions,*[12] Thomas Kuhn points out that a scientific paradigm can consist of a set of solved problems. Reusable software project artifacts are a set of solved problems—solved requirements problems, design problems, planning problems, management problems, and so on.

The Call of Engineering

Arthur C. Clarke said that "any sufficiently advanced technology is indistinguishable from magic." Software technology is sufficiently advanced, and the general public is mystified by it. The public doesn't understand the safety risks posed by its software products or the financial risks posed by its software projects. As high priests of this powerful magic, software developers have a professional responsibility to use their magic wisely.

The engineering approach to design and construction has a track record of all but eliminating some of the most serious risks to public safety while supporting some of the most elevating expressions of the human spirit. Whether the goal is safety, aesthetics, or economics, treating software as an engineering discipline is an effective way to raise software development to the level of a true profession.

Notes

1. Florman, Samuel C., *The Existential Pleasures of Engineering,* 2d Ed., NY: St. Martin's Griffin, 1994.
2. I call this "folklore" because several Canadian professional engineers have independently told me that the iron in the iron ring traditionally is thought to come from the wreckage of the iron bridge that collapsed in Quebec City in 1907. Published information about the Canadian iron ring ceremony contains no mention of the Quebec City bridge. The ceremony itself might contain mention of this bridge, but it is a secret ceremony.
3. *The New York Times,* May 3, 1999.
4. Digest subscription to this forum is available by e-mailing risks-request@csl.sri.com or on Usenet at comp.risks.
5. All the examples in this paragraph come from Peter G. Neumann, *Computer-Related Risks,* Reading, MA: Addison-Wesley, 1995.
6. This image was obtained from IMSI's MasterClips©/MasterPhotos© collection, 1895 Francisco Blvd. East, San Rafael, CA 94901-5506, USA.
7. This image was obtained from IMSI's MasterClips©/MasterPhotos© collection, 1895 Francisco Blvd. East, San Rafael, CA 94901-5506, USA.

8. Shaw, Mary, "Prospects for an Engineering Discipline of Software," *IEEE Software,* November 1990, pp. 15f.
9. Christopher Alexander, quoted in Glass, Robert L., *Software Creativity,* Englewood Cliffs, NJ: Prentice Hall PTR, 1994.
10. Simon, Herbert, *The Sciences of the Artificial,* 3d Ed., Cambridge, MA: MIT Press, 1996.
11. For more information, see Jones, Capers, *Assessment and Control of Software Risks,* Englewood Cliffs, NJ: Yourdon Press, 1994.
12. Kuhn, Thomas S., *The Structure of Scientific Revolutions,* 3d Ed., Chicago: The University of Chicago Press, 1996.