Chapter 1

The Scope of Software Engineering

Learning Objectives

After studying this chapter, you should be able to

- Define what is meant by software engineering.
- Describe the classical software engineering life-cycle model.
- Explain why the object-oriented paradigm is now so widely accepted.
- Discuss the implications of the various aspects of software engineering.
- Distinguish between the classical and modern views of maintenance.
- Discuss the importance of continual planning, testing, and documentation.
- Appreciate the importance of adhering to a code of ethics.

A well-known story tells of an executive who received a computer-generated bill for $0.00. After having a good laugh with friends about "idiot computers," the executive tossed the bill away. A month later, a similar bill arrived, this time marked 30 days. Then came the third bill. The fourth bill arrived a month later, accompanied by a message hinting at possible legal action if the bill for $0.00 was not paid at once.

The fifth bill, marked 120 days, did not hint at anything—the message was rude and forthright, threatening all manner of legal actions if the bill was not immediately paid. Fearful of his organization's credit rating in the hands of this maniacal machine, the executive called an acquaintance who was a software engineer and related the whole sorry story. Trying not to laugh, the software engineer told the executive to mail a check for $0.00. This had the desired effect, and a receipt for $0.00 was received a few days later. The executive meticulously filed it away in case at some future date the computer might allege that $0.00 was still owed.

This well-known story has a less well-known sequel. A few days later, the executive was summoned by his bank manager. The banker held up a check and asked, "Is this your check?"

The executive agreed that it was.

"Would you mind telling me why you wrote a check for $0.00?" asked the banker.

So the whole story was retold. When the executive had finished, the banker turned to him and she quietly asked, "Have you any idea what your check for $0.00 did to *our* computer system?"

A computer professional can laugh at this story, albeit somewhat nervously. After all, every one of us has designed or implemented a product that, in its original form, would have resulted in the equivalent of sending dunning letters for $0.00. Up to now, we have always caught this sort of fault during testing. But our laughter has a hollow ring to it, because at the back of our minds is the fear that someday we will not detect the fault before the product is delivered to the customer.

A decidedly less humorous software fault was detected on November 9, 1979. The Strategic Air Command had an alert scramble when the worldwide military command and control system (WWMCCS) computer network reported that the Soviet Union had launched missiles aimed toward the United States [Neumann, 1980]. What actually happened was that a simulated attack was interpreted as the real thing, just as in the movie *WarGames* some 5 years later. Although the U.S. Department of Defense understandably has not given details about the precise mechanism by which test data were taken for actual data, it seems reasonable to ascribe the problem to a software fault. Either the system as a whole was not designed to differentiate between simulations and reality or the user interface did not include the necessary checks for ensuring that end users of the system would be able to distinguish fact from fiction. In other words, a software fault, if indeed the problem was caused by software, could have brought civilization as we know it to an unpleasant and abrupt end. (See Just in Case You Wanted to Know Box 1.1 for information on disasters caused by other software faults.)

Whether we are dealing with billing or air defense, much of our software is delivered late, over budget, and with residual faults, and does not meet the client's needs. Software engineering is an attempt to solve these problems. In other words, **software engineering** is a discipline whose aim is the production of fault-free software, delivered on time and within budget, that satisfies the client's needs. Furthermore, the software must be easy to modify when the user's needs change.

The scope of software engineering is extremely broad. Some aspects of software engineering can be categorized as mathematics or computer science; other aspects fall into the areas of economics, management, or psychology. To display the wide-reaching realm of software engineering, we now examine five different aspects.

1.1 Historical Aspects

It is a fact that electric power generators fail, but far less frequently than payroll products. Bridges sometimes collapse but considerably less often than operating systems. In the belief that software design, implementation, and maintenance could be put on the same