More than 200 years after George Washington was elected the first U.S. president, the November 2004 presidential election will usher in a new experiment in democracy—the first widespread use of electronic voting technology in a national election. Whatever your attitude about e-voting systems and whatever you may have read about them in recent months, they are an example of the software-intensive systems that every industrialized society increasingly depends on for their basic operations. Software-intensive systems help protect and manage commercial air travel, operate the electric power grid, file tax returns, and may one day even uphold our democratic foundations.

Yet today, in spite of more than three decades of effort, we still lack what could be called a solid scientific basis for building any kind of large-scale system that includes software elements. Our software-intensive systems are so complex we often find ourselves struggling to understand and control them. New systems are created and adopted with high hopes, crossed fingers, and even controversy, as in the case of e-voting. Security breaches, privacy violations, and emergent (unexpected) properties are some of the regular, often unwelcome, results.

For computer scientists, being told that creating software-intensive systems is difficult is not news. Even attendees at the 1968 NATO Software Engineering Conference in Garmisch, Germany, recognized the need to build, for example, a foundation for the systematic creation of software-intensive systems [3]. Efforts have continued since then to develop a scientific basis and process for creating them.

Living in a world in which the number and diversity of devices, amount of software, and degree of connectivity in complex systems are all increasing by orders of magnitude, it is essential that we have a science of design on which to base our efforts to create these systems. Changing this situation won’t happen overnight, but we also can’t wait another three decades to begin in earnest.

While it’s easy to assert the need for a science of design, defining what such a science entails and understanding what design really means are much greater challenges.

By way of analogy, the design of a large building is based on many scientifically discovered and validated facts, along with volumes of codified experience. A similar scientific basis exists for many engineered artifacts, ranging from integrated circuits to airplanes. Professional disciplines like architecture and engineering have been working on issues of design for far longer than a few decades. In general, they have a stronger scientific footing, and computer scientists can learn where that footing overlaps with computing. (The National Science Foundation has funded at least one ongoing project to research how information technology might better assist the architectural design process.)

On the other hand, software-intensive systems are complex, being composed of software, people, computers, and other devices. Though the other components’ connections to the software and their role in the overall design of the system are critical, the core consideration for a software-intensive system is the software itself, and other approaches to systematizing design have yet to solve the “software problem”—which won’t be solved until software
design is understood scientifically.

A more complete definition of a science of design may be found through close examination of the words “science” and “design.” Here, design encompasses all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems—not just the activity following requirements specification and before programming, as it might be translated from a stylized software engineering process.

Design is the central issue when dealing with artifacts, or objects created by people, as opposed to those occurring naturally. Design is critical, no matter what is done with the artifact. Specifying an artifact means you are shaping the design to meet its requirements. If you build or implement an artifact, the design is your guide. If you modify an artifact, you must deal with its design at many levels [2].

A dictionary might define science as “the observation, identification, description, experimental investigation, and theoretical explanation of natural phenomena” [1]. Science methodically seeks to understand the principles underlying these phenomena. Science also commonly refers to a body of knowledge that is intellectually rigorous, analytical, formalized where appropriate, based on empirical studies where possible, and, above all, teachable to future generations of scientists.

By juxtaposing these two ideas in a science of design, we begin to see the broad outlines of our challenge. We need an intellectually rigorous, formalized, and teachable body of knowledge about the principles underlying software-intensive systems and the processes used to create them. The centrality of design would suggest that many other essential principles would become easier to obtain once we have a solid basis for design. We must also be willing to expand the concept of science from natural to artificial phenomena, though we in computer science, at least, have long since come to grips with that concept.

What might a science of design include? To borrow from the thinking of the late Herbert A. Simon, as he described it in his book *The Sciences of the Artificial*, design might be dissected into phases that include the generation and selection of design alternatives, design representations, solution procedures for design problems, human approaches to design, and design structures [4].

The first of these phases might be modeled as, say, a process for choosing among alternative structures that make up the design. Questions to be answered include: How can alternative designs be generated for consideration? How can their properties be accurately described? How can we order the decision-making process to minimize the time needed to obtain a design that best satisfies the desired criteria?

Alternative models of the design process may lead to yet other sets of questions. Ultimately, having a true science of design means we’ll know what those different models are and have information about their validity, as well as about the questions that flow from each of them.

But design cannot be boiled down to an antiseptic exercise in logic and optimization. A science of design must also account for such factors as style and innovation, reaching beyond the requirements for basic functionality to the context in which software-intensive systems exist. How might e-voting systems be designed, for example, to protect civil rights and personal privacy while minimizing political bias and security threats? How might urban planning systems be designed to account for social and cultural diversity, public decision making, and accessibility? How might systems using pervasive “disposable” technologies (such as wireless sensor networks) be designed to avoid adverse effects on people’s physical health, personal privacy, and civil liberties?

Perhaps the most important question of all is: How do we ensure that we learn from our failures?
and justifiably boasts many successes, steam engines have exploded, bridges collapsed, buildings burned, and flying machines crashed—not to mention other, less dramatic failures.

Many engineered artifacts (such as bridges and canals) are public works, whose failures are subject to public examination. Others (such as steam engines) are proprietary, with the knowledge kept secret within the limited scope of companies. Because so much software is proprietary, even in publicly deployed systems, a special challenge for the science of design is to find ways to learn from our failures, spectacular or otherwise, as we build software-intensive systems.

NSF has sponsored several workshops to investigate aspects of this important topic, and design is a key research focus in many of the programs in the NSF’s Computer and Information Science and Engineering directorate, as well as in its Engineering directorate. NSF will also soon be announcing awards from this year’s Science of Design competition; see the program solicitation at www.cise.nsf.gov/funding/pgm_display.cfm?pub_id=13078.

Creating a science of design requires computer scientists and engineers in academic institutions and in industry to think about software differently and make fundamental changes in the way they work. But without a science of design, it is likely that we will soon be unable to deal with the opportunities before us.

References

Peter Freeman (pfreeman@nsf.gov) is the assistant director for Computer and Information Science and Engineering at the National Science Foundation in Arlington, VA.

David Hart (dahart@nsf.gov) is an information specialist in the National Science Foundation’s Office of Legislative and Public Affairs in Arlington, VA.