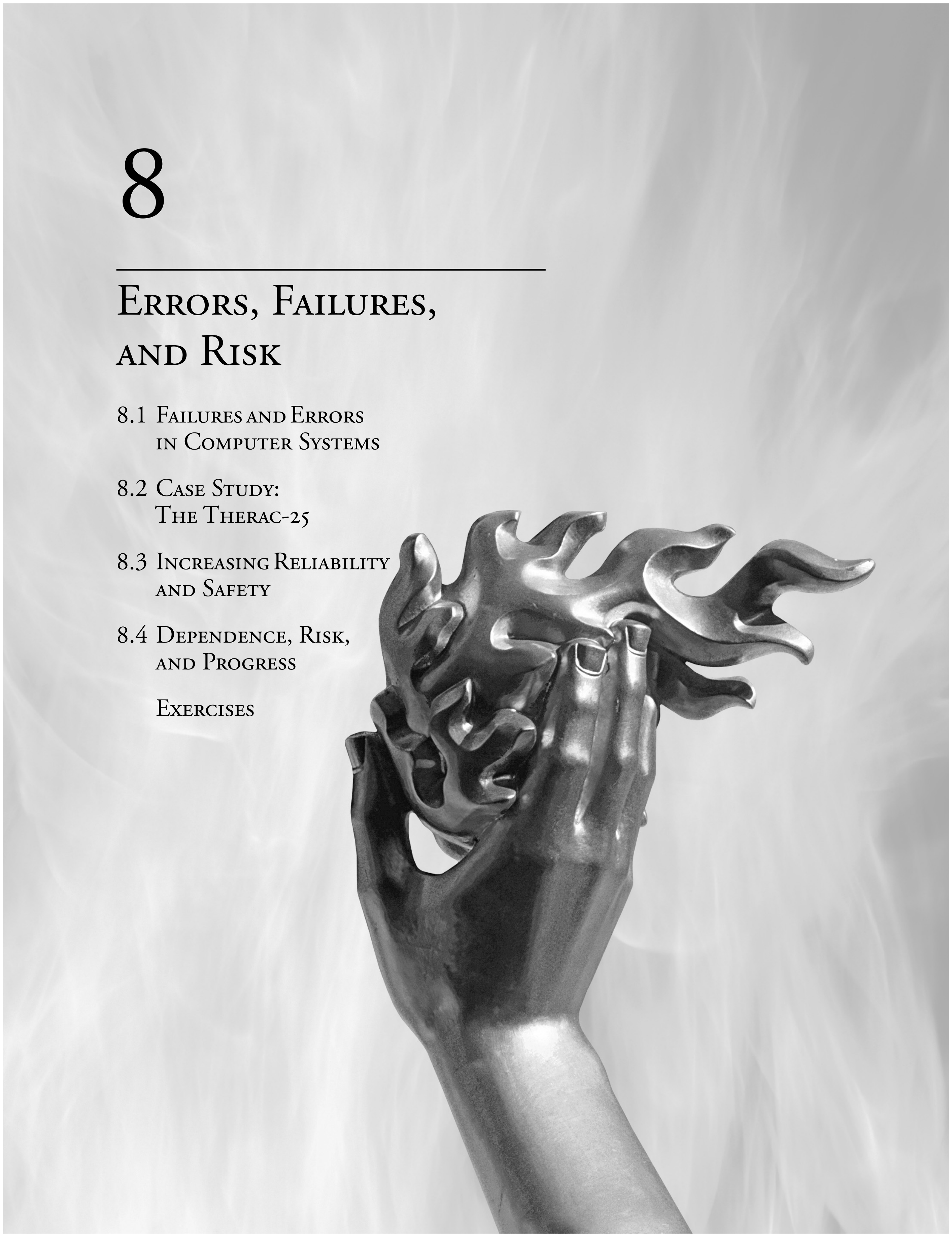
8 Errors, Failures, and Risk



Most of this chapter is about computer systems that fail, crash, or never work at all because of software bugs, poor design, political considerations, or other factors. Some systems work as intended, but serious damage results when people misinterpret or misuse the information they provide. Errors in databases inconvenience people and disrupt lives. We consider a physical risk of computer use also: hand and arm problems. Studying these failures and risks contributes to understanding their causes and helps prevent future failures.

8.1 Failures and Errors in Computer Systems

8.1.1 AN OVERVIEW

"Navigation System Directs Car into River"
"Data Entry Typo Mutes Millions of U.S. Pagers"
"Flaws Found in Software that Tracks Nuclear Materials"
"Software Glitch Makes Scooter Wheels Suddenly Reverse Direction"
"IRS Computer Sends Bill for $68 Billion in Penalties"
"Robot Kills Worker"
"California Junks $100 Million Child Support System"
"Man Arrested Five Times Due to Faulty FBI Computer Data"

These headlines describe real incidents. Most computer applications, from consumer software to systems that control communications networks, are so complex that it is virtually impossible to produce programs with no errors. In the next few sections, we describe a variety of mistakes, problems, and failures—and some factors responsible for them. Some errors are minor. For example, a word processor might incorrectly hyphenate a word that does not fit at the end of a line. Some incidents are funny. Some are tragic. Some cost billions of dollars. All the examples can teach us something.

Are computer-controlled medical devices, factory automation systems, and airplane flight-management systems too unsafe to use? Or, like many stories on the evening news, do the headlines and horror stories emphasize the bad news—the dramatic and unusual events? We hear reports of car crashes on the news, but we do not hear that drivers completed 200,000 car trips safely in our city today. Although most car trips are safe, there is a good purpose for reporting crashes on the news: It teaches us what the risks are (e.g., driving in heavy fog) and it reminds us to be responsible and careful drivers. Just as many factors cause car crashes (faulty design, sloppy manufacturing or servicing, bad road conditions, a careless or poorly trained driver, confusing road signs, and so on), computer glitches and system failures also have myriad causes, including faulty design, sloppy implementation, careless or insufficiently trained users, and poor user interfaces. Often, there is more than one factor. Because of the complexity of computer systems, it is essential to follow good procedures and professional practices for their development and use. Sometimes, no one does anything clearly wrong, but an accident occurs anyway. Occasionally, the irresponsibility of software developers and managers is comparable to driving while very drunk.

Although millions of computers and software programs are working fine every day, it is crucial that we understand the risks and reasons for computer failures. How much risk must or should we accept? If the inherent complexity of computer systems means they will not be perfect, how can we distinguish between errors we should accept as trade-offs for the benefits of the system and errors that are due to inexcusable carelessness, incompetence, or dishonesty? How good is good enough? When should we, or the government, or a business decide that a computer is too risky to use? Why do multimillion-dollar systems fail so miserably that the firms and agencies that pay for them abandon them before completion? We cannot answer these questions completely, but this chapter provides some background and discussion that can help us in forming conclusions. It should help us understand computer-related problems from the perspective of several of the roles we play:

- *A computer user.* Whether we use a personal computer or a sophisticated, specialized system at work, we should understand the limitations of computers and the need for proper training and responsible use. We must recognize that, as in other areas, there are good products and bad products.
- *A computer professional.* Studying computer failures should help you become a better computer professional (system designer, programmer, or quality assurance manager, for example) if that is your career direction. Understanding the source and consequences of computer failures is also valuable if you will be responsible for buying, developing, or managing a complex system for a hospital, airport, or business. The discussions of the examples in this chapter include many implicit and explicit lessons about how you can avoid similar problems.
- *An educated member of society.* There are many personal decisions and social, legal, and political decisions that depend on our understanding of the risks of computer system failures. We could be on a jury. We could be an active member of an organization lobbying for legislation. We could be deciding whether or not to try an experimental computer-controlled medical device. Also we can apply some of the problem-solving approaches and principles in this chapter to professional areas other than computer systems.

We can categorize computer errors and failures in several ways, for example, by the cause, by the seriousness of the effects, or by the application area. In any scheme to organize the discussion, there will be overlap in some categories and mixing of diverse examples in some. For the remainder of this section, I use three categories: problems for individuals, usually in their roles as consumers; system failures that affect large numbers of people and/or cost large amounts of money; and problems in safety-critical applications that may injure or kill people. We will look at one case in depth (in Section 8.2): the Therac-25. This computer-controlled radiation treatment machine had a large number of flaws that resulted in the deaths of several patients. In Sections 8.3 and 8.4, we try to make some sense of the jumble of examples. Section 8.3 looks at underlying causes in more depth and describes some approaches to reducing problems. Section 8.4 puts the risks of computer systems into perspective in various ways, including considering risks in other systems and risks due to not using computers.

The incidents described here are a sampling of the many that occur. Robert Charette, an expert on software risk management, emphasizes that computer system errors and failures occur in all countries, in systems developed for businesses, governments, and nonprofit organizations (large and small) "without regard to status or reputation."[1] In most cases, by mentioning specific companies or products, I do not mean to single those out as unusual offenders. One can find many similar stories in news reports, in software engineering journals, and especially in the *Risks Digest* organized by Peter Neumann.[2] Neumann collects thousands of reports describing a wide range of computer-related problems.

8.1.2 PROBLEMS FOR INDIVIDUALS

Many people are inconvenienced and/or suffer losses from errors in billing systems and databases containing personal data.

Billing errors

The first few errors we look at are relatively simple ones whose negative consequences were undone relatively easily.[3]

- A woman received a $6.3 million bill for electricity. The correct amount was $63. The cause was an input error made by someone using a new computer system.
- The IRS is a constant source of major bloopers. When it modified its programs to avoid billing victims of a Midwest flood, the computer generated erroneous bills for almost 5,000 people. One Illinois couple received a bill for a few thousand dollars in taxes—and $68 billion in penalties. In one year, the IRS sent 3,000 people bills for slightly more than $300 million. One woman received a tax bill for $40,000,001,541.13.
- The auto insurance rate of a 101-year-old man suddenly tripled. Rates depend on age, but the program handled ages only up to 100. It mistakenly classified the man as a teenager.
- Hundreds of Chicago cat owners received bills from the city for failure to register dachshunds, which they did not own. The city used computer matching with two databases to try to find unlicensed pets. One database used DHC as the code for domestic house cat, and the other used the same code for dachshund.

Programmers and users could have avoided some of these errors. For example, programmers can include tests to determine whether a billing amount is outside some reasonable range or changed significantly from previous bills. In other words, because programs can contain errors, good systems have provisions for checking their results. If you have some programming experience, you know how easy it would be to include such tests and make a list of cases for someone to review. These errors are perhaps more humorous than serious. Big mistakes are obvious. They usually get fixed quickly. They are worth studying because the same kinds of design and programming errors can have more serious consequences in different applications. In the Therac-25 case (Section 8.2) we will see that including tests for inconsistent or inappropriate input could have saved lives.

Inaccurate and misinterpreted data in databases

Credit bureau records incorrectly listed thousands of New England residents as not having paid their local property taxes. An input error appeared to be the cause of the problem. People were denied loans before someone identified the scope of the problem and the credit bureau corrected it. (The credit bureau company paid damages to many of the people affected.) Like $40-billion tax bills, a systematic error affecting thousands of people is likely to get noticed. The relevant company or agency is likely to fix it quickly. More serious perhaps are all the errors in individual people's records. Critics of credit bureaus argue that incorrect information in credit records cause people to lose homes, cars, jobs, or insurance. In one case, a county agency used the wrong middle name in a report to a credit bureau about a father who did not make his child-support payments. Another man in the same county had the exact name reported. He could not get credit to buy a car or a house. It is difficult to get accurate and meaningful error rates. We need to distinguish between a spelling error in someone's address and an incorrect report that someone bounced several checks. The results of numerous surveys and studies vary considerably, but all indicate that a high percentage of credit records have serious errors. Many people battle for years to get the credit bureaus to correct information in their records, and a few have won large settlements in lawsuits.

Errors in background checks of employees: Section 6.2.4

Federal law requires states to maintain databases of people convicted of sex crimes against children and to release information about them to the public. A family was harassed, threatened, and physically attacked after its state posted an online list of addresses where sex offenders live. The state did not know the offender had moved away before the family moved in. A lawyer reported that a man was nearly beaten to death after being mistaken for his brother. The brother was a sex offender in New Jersey's database. A man murdered two sex offenders in Washington state after getting their addresses from the state's sex offender database, and another man killed two sex offenders listed in Maine's online registry. There was no indication of database errors in these cases, but they clearly illustrate the risks. A man convicted of statutory rape for having sex with his 17-year-old girlfriend, and now married to her for many years, was in a sex offender database. While technically not an error, this case illustrates the need for careful thought about what a database includes and how it is presented to the public, especially if it involves a highly charged subject.

Florida voting officials mistakenly purged many people from its voting rolls. They could not vote in the close 2000 U.S. presidential election because their names matched names of convicted felons (who lose their voting rights). The state had hired a division of ChoicePoint, a huge database company, to supply lists of convicted felons. Apparently, the election officials used the lists without the extra verification step the company warned they would need.[4]

A high school excluded a 14-year-old boy from football and some classes without explanation. He eventually learned that school officials thought he had been using drugs while in junior high school. The two schools used different disciplinary codes in their computerized records. The boy had been guilty of chewing gum and being late. This case is very similar to the case of the dachshund/cat confusion described earlier—except that the consequences were more significant. Both cases illustrate the problems of relying on computer systems without taking the responsibility of learning enough about them to use them properly.

When errors occur in databases used by law enforcement agencies, the consequences can include arrests at gunpoint, strip searches, and time in jail with violent criminals. For example, two adults went to jail and a child to a juvenile home for 24 hours while police determined that they had really rented the rental car they were driving. The car rental company had listed the car as stolen. Studies of the FBI's National Crime Information Center (NCIC) database found that a high percentage of the arrest warrants in it were inaccurate or no longer valid. People are arrested when a check of the database shows a warrant for them—or for someone with a similar name. I will mention a few NCIC cases and other law enforcement cases. The news media and government studies reported many more. An adoption agency ran a routine check on an applicant and found a conviction for grand larceny. In fact, the applicant took part in a college prank—stealing a restaurant sign—years before. He had apologized and paid for the damage, and the charges had been dropped. The error could have caused the agency to deny the adoption. Police arrested a Michigan man for several crimes, including murders, committed in Los Angeles. Another man had assumed his identity after finding his lost wallet (or a discarded birth certificate; reports varied). It is understandable that NCIC listed the innocent man as wanted. However, he was arrested four more times within 14 months. (After repeatedly asking the city of Los Angeles to correct the records, he sued and won a judgment against the city.) The military imprisoned a man for five months because NCIC mistakenly reported that he was AWOL.* A college professor returning from London spent two days in jail after a routine check with NCIC at Customs showed that he was a wanted fugitive. NCIC was wrong—for the third time about this particular man. Police stopped and frisked an innocent driver because his license plate number incorrectly appeared as

*AWOL means "absent without official leave."

the license number of a man who had killed a state trooper. The computer record did not include a description of the car. (NCIC now includes digitized photographs and fingerprints to help reduce the number of incidents in which police detain an innocent person.)[5]

After September 11, 2001, the FBI gave a "watch list" to police departments and businesses such as car rental agencies, banks, casinos, and trucking and chemical firms. Recipients e-mailed the list to others, and eventually thousands of police departments and thousands of companies had copies. Many incorporated the list into their databases and systems that screened customers or job applicants. Although the list included people who were not suspects but whom the FBI wanted to question, some companies labeled the list "Suspected terrorists." Many entries did not include date of birth, address, or other identifying information, making mistaken identifications likely. Some companies received the list by fax and typed misspelled names from blurred copies into their databases. The FBI stopped updating the list but did not tell the recipients; thus many entries became obsolete.[6]

According to the Transportation Security Administration, more than 30,000 people have been mistakenly matched to names on terrorist watch lists at airports and border crossings. (The agency established a procedure to create a "cleared list" for such people so that they will not be stopped repeatedly in the future.)[7]

Several factors contribute to the frequency and severity of the problems people suffer because of errors in databases and misinterpretation of their contents:

- a large population (many people have identical or similar names, and most of our interactions are with strangers)
- automated processing without human common sense or the power to recognize special cases
- overconfidence in the accuracy of data stored on computers
- errors (some because of carelessness) in data entry
- failure to update information and correct errors
- lack of accountability for errors.

The first item is unlikely to change. It is the context in which we live. The second is partly a side effect of the speed and processing ability of computer technology, but we can reduce its negative impacts with better system specifications and training of users. Even if someone corrects an error in a database, problems may not be over for the affected person. Computer records are copied easily and often. Copies of the incorrect data may remain in other systems. The remaining factors in the list above are all within our control as individuals, professionals, and policy makers. We discuss some solutions in Section 8.3.

It is repugnant to the principles of a free society that a person should ever be taken into police custody because of a computer error precipitated by government carelessness. As automation increasingly invades modern life, the potential for Orwellian mischief grows.

—Arizona Supreme Court[8]

If you're trying to type War and Peace *with your thumbs, then you're going to have a problem.*

—Alan Hedge, director of the Human Factors and Ergonomics Lab, Cornell University[9]

BLACKBERRY THUMB AND RSI

Millions of children play games on small electronic devices, and millions of adults answer e-mail on portable electronic gadgets with mini keypads. In many professions, people type on a computer keyboard for hours each day. Most of the risks we describe in this chapter result from errors in software, poor system design, or inaccurate and misinterpreted information. Here, we look at physical phenomena known as BlackBerry Thumb, Gamer's Thumb, Nintendonitis, repetitive strain injury (RSI), and various other terms. Repetitive strain injury, the more formal term, covers a variety of injuries or pain in thumbs, fingers, wrists, and arms (and sometimes neck and shoulders). You may have seen computer programmers, prolific bloggers, secretaries, or supermarket checkers (who move products quickly past a bar-code scanner for hours) wearing wrist braces, called splints—a common sign of RSI. These injuries can make ordinary activities painful or impossible and can prevent people from working.

RSI is not a new disease. There are references to similar problems in the 18th and 19th-centuries afflicting clerks and scribes (we used to call this writer's cramp), women who milked cows, and others whose work required repetitive hand motions. RSI problems occur among gymnasts, sign-language interpreters for the deaf, "pushup enthusiasts," auto workers, seamstresses, musicians, carpenters, meat processors, and workers in bakery factories. (An article in the *Journal of the American Medical Association* listed 29 occupations with common RSI problems.)[10] Computer game players, cell phone and keyboard users, and users of personal digital assistants (the BlackBerry is just one example) are among the newest significant group of RSI sufferers.

Thousands of people suffering from RSI sued keyboard makers and employers in the 1990s. They charged that the companies were at fault and should pay for medical costs and damages to the victims. Many of the suits resulted in dismissals or decisions for the defendants. In a few cases where plaintiffs won large awards, higher courts overturned the decisions on appeal. The uncertainty of causation (defects in the devices or improper use) made it difficult to win such suits. Some judges and others compare the complaints to ordinary aches and pains from overexercising or overusing a normally safe tool or device. What would we think of an RSI lawsuit against the maker of a tennis racket or a violin?

Attention to proper ergonomic design of keyboards and workstations reduced RSI problems for keyboard users. Laptop computer makers redesigned the machines to include a wrist rest. We can now buy split, twisted, and otherwise nontraditionally shaped keyboards—each one implementing some manufacturer's idea of what will be more comfortable and reduce strain. Modifying equipment alone does not solve the problem. RSI experts stress the importance of training in proper technique (including the importance of rest breaks, posture, and exercises). One can install free software that interrupts the user at regular intervals for rest breaks and software-guided exercises. Speech input devices might also reduce RSI caused by keyboard use. (But we might discover an increase in strain of the vocal cords.) Partly because of growing recognition of the RSI problem, and partly as protection against lawsuits, computer companies now provide information about proper use and arrangement of keyboards. Some game device makers package their product with reminders for users to take rest breaks.

Adult users of any tool or toy should learn proper techniques for its use. Young children need parental supervision or rules, as they do, for example, about wearing a helmet when riding a bicycle. Employers have a responsibility to provide training in proper and safe use of tools. Being aware of the potential for RSI might or might not encourage game players and instant messagers to take breaks and rest their hands and fingers. Mothers and doctors tell us repeatedly that we should sit up straight, exercise often, and eat our vegetables. Many people do not follow this advice—but, once we have the information, we can choose what to do with it.

Balance is very important for hand comfort. You'll be surprised at how quick your wrist will ache if the knife is not balanced properly.

—George McNeill, Executive Chef, Royal York Hotel, Toronto (on an advertisement for fine cutlery)

8.1.3 SYSTEM FAILURES

Modern communications, power, medical, financial, retail, and transportation systems depend heavily on computer systems. They do not always function as planned. We describe a lot of failures, some with indications of the causes. For computer science students and others who might contract for or manage custom software, one aim is to see the serious impacts of the failures—and to see what you want to work hard to avoid. The lessons of adequate planning, of making backup plans in case of failures, and of responsibility apply to large projects in other professions as well.

Communications, business, and transportation

Customers of AT&T lost telephone service for voice and data for nine hours because of a software error in a four-million line program. The disruption prevented roughly 50 million calls from getting through. A three-line change in a two-million line telecommunications switching program caused a failure of telephone networks in several major East Coast and West Coast cities. Although the program underwent 13 weeks of testing, it was not retested after the change—which contained a typo. A glitch in a routine software upgrade at America Online prevented subscribers from logging in all over the U.S. for several hours. American Express Company's credit-card verification system failed during the Christmas shopping season. Merchants had to call in for verification, overwhelming the call center. A majority of Skype's Internet phone users could not log in for two days in 2007. Its peer-to-peer network system had become overloaded by log-ins when a huge number of people rebooted their computers after installing routine Windows updates. (Skype has roughly 220 million users.)

When a Galaxy IV satellite computer failed, many systems we take for granted stopped working. Pager service stopped for an estimated 85% of users in the U.S., including hospitals and police departments. The failure interrupted radio and television broadcasts. Airlines that got their weather information from the satellite had to delay flights. The gas stations of a major chain could not verify credit cards. Some services were quickly switched to other satellites or backup systems. It took days to restore others.[11]

Every few years, the computer system of one of the world's largest stock exchanges or brokerages fails. An error in a software upgrade shut down trading on the Tokyo Stock Exchange. A problem in new communications software virtually shut down the NASDAQ stock exchange for two and a half hours. A glitch in an upgrade in the computer system at Charles Schwab Corporation crashed the system for more than two hours and caused intermittent problems for several days. Customers could not access their accounts or trade online. A computer malfunction froze the London Stock Exchange for almost eight hours—on the last day of the tax year, affecting many people's tax bills.[12]

A failure of Amtrak's reservation and ticketing system during Thanksgiving weekend caused delays because agents had no printed schedules or fare lists. Two large travel reservation systems that handle reservations for airlines, car rental companies, and hotels shut down for many hours because of computer problems. American Airlines could not verify electronic tickets; it delayed 200 flights. A failure of the computer that prepares flight plans for America West Airlines delayed thousands of passengers. AirTran installed a new system to handle flight check-in on the Internet, at airport self-service kiosks, and at airport check-in counters. It failed on its first day. Passengers and ticket agents could not print boarding passes; many people missed flights. Sometimes systems fail because they attempt something radically new. The AirTran failure, however, occurred in 2006, after air travelers had been checking in online and at self-service kiosks for several years.

The $125-million Mars Climate Orbiter disappeared when it should have gone into orbit around Mars. One team working on the navigation software used English measure units while another team used metric units. The investigation of the loss emphasized that while the error itself was the immediate cause, the fundamental problem was the lack of procedures that would have detected the error.[13]

Destroying businesses

Several companies have gone bankrupt after spending a huge amount of money on computer systems that failed to work. We describe one case of a system that seriously strained some businesses.

A few dozen companies that bought an inventory system called Warehouse Manager blamed the system for disastrous losses. One previously successful company saw its income decline by about half and laid off half its employees. The specific complaints were numerous. One company could not get the system to place a purchase order for several weeks. The company claimed the backlog in orders cost $2,000 per day. Processes that should have taken seconds, such as printing invoices, took several minutes. Impatient customers waited in long lines. The system gave incorrect information about inventory. It informed clerks that products were in stock when they were not and vice versa. According to users of Warehouse Manager, the system reported incorrect prices to clerks. A part that cost $114 was listed for sale at 54 cents. A $17 part was listed for sale at $30. The first error means lost money for the company. The second means lost customers—those who go elsewhere to find a better price. When two clerks tried to access the computer from their terminals simultaneously, the terminals locked up. Some companies said the system erased information needed for accounting and tax reports.[14]

What was responsible for the problems in Warehouse Manager? NCR Corporation sold the program, but another company developed it. The company originally designed and implemented the program on a different computer and operating system. It appears that there were unexpected problems when the company rewrote the program for NCR's machines and its ITX operating system. According to the *Wall Street Journal*, internal memos at NCR reported inadequate testing and poor performance in real business settings. NCR salespeople told prospective customers that Warehouse Manager was running successfully at 200 installations, but most of those were installations using the machine for which the program was originally designed. Several users claimed that although NCR was receiving complaints of serious problems from many customers,

DESTROYING CAREERS AND SUMMER VACATIONS[15]

CTB/McGraw-Hill develops and scores standardized tests for schools. About nine million students take its tests each year. An error in CTB's software caused it to report test results incorrectly—substantially lower than the correct scores—in several states. In New York City, school principals and superintendents lost their jobs because their schools appeared to be doing a poor job of teaching students to read. Educators endured personal and professional disgrace. One man said he applied for 30 other superintendent jobs in the state but did not get one. Parents were upset. Nearly 9,000 students had to attend summer school because of the incorrect scores. Eventually, CTB corrected the error. New York City's reading scores had actually risen five percentage points.

Why was the problem not detected sooner, soon enough to avoid firings and summer school? School testing officials in several states were skeptical of the scores showing sudden, unexpected drops. They questioned CTB, but CTB told them nothing was wrong. They said CTB did not tell them that other states experienced similar problems and also complained. When CTB discovered the software error, the company did not inform the schools for many weeks, even though the president of CTB met with school officials about the problem during those weeks.

What lessons can we learn from this case? Software errors happen, of course. People usually notice significant mistakes, and they did here. But the company did not take seriously enough the questions about the accuracy of the results and was reluctant to admit the possibility—and later the certainty—of errors. It is this behavior that must change. The damage from a computer error can be small if the error is found and corrected quickly.

CTB recommended that school districts not use scores on its standardized tests as the sole factor in deciding which students should attend summer school. But New York City did so. We saw earlier that Florida state officials relied on computer-generated lists to prevent some people from voting, even though the database company supplying the lists said they needed additional verification. Relying solely on results produced by computers is temptingly easy. It is a temptation that people responsible for critical decisions in many situations should resist.

the company told them the problems they were having were unique. NCR blamed the problems on the company that developed Warehouse Manager and modified it for ITX. Eventually, NCR agreed it "did not service customers well" and the program should have undergone more extensive testing. The company settled most of the few dozen lawsuits out of court, with confidentiality agreements about the terms. The sources of the problems in this case included technical difficulties (converting software to a different system), poor management decisions (inadequate testing), and, according to the customers, dishonesty in promoting the system and responding to the problems.

Voting systems

The U.S. presidential election of 2000 demonstrated some of the problems of old-fashioned election machines and paper or punch-card ballots. Vote counters found these ballots sometimes difficult to read or ambiguous. Recounting was a slow tedious process. In 2002, Congress passed the Help America Vote Act and authorized $3.8 billion to improve voting systems. Many saw electronic systems, some using touch screens, as the solution. By the 2006 elections, a very small percent of Americans still voted with paper ballots. The rush to electronic voting machines demonstrated that they too could have numerous faults. We consider some of the problems in elections of 2002–2006.

Some electronic voting systems just crashed—voters were unable to vote. Machines in North Carolina failed to count more than 400 votes because of a technical problem. One county lost more than 4,000 votes because the machine's memory was full. A programming error generated 100,000 extra votes in a Texas county. A programming error caused some candidates to receive votes actually cast for other candidates.

Security against vote fraud and sabotage is another worry. Programmers or hackers can intentionally rig software to give inaccurate results. Depending on the structure of the system, independent recounting may be difficult. Security researchers strongly criticized voting machines made by the company that supplies a large portion of the machines. They say the machines have insecure encryption techniques, insufficient security for installation of upgrades to software, and poor physical protection of the memory card on which the system stores votes. Researchers opened the access panel on a voting machine with a standard key that is easily available and used in office furniture, electronic equipment, and hotel minibars.[16]

Independent groups of people can recount paper ballots, so more than half the states passed laws requiring some kind of paper record of votes. Poll workers in one county loaded the paper incorrectly and lost about 10% of the votes.

In some counties, election officials gave voting machines to high-school students and other volunteers to store at home and deliver to polling places on election day.

Many of the failures that occurred result from causes we will see over and over: lack of sufficient planning and thought about security issues, insufficient testing, and insufficient training. (In this application, the training task is complex. Thousands of ordinary people volunteer as poll workers and manage and operate the machines on election day.) In projects like these, the desire of states to obtain federal grants sometimes encourages haste.

One policy that can help to reduce errors and fraud is to make the software in electronic voting machines public. Then various experts and tech-savvy members of the public can examine it. Companies that produce electronic voting systems have resisted this, arguing that the software is proprietary. If they release it, they might lose an advantage over their competitors. Of course, even if the published version of the software is correct, secure, and free of intentional vote manipulation, we need procedures to ensure that the actual machines used in the voting process do not have different software installed.

Long before we voted on computers, Chicago and parts of Texas were infamous for vote fraud. In some cities, election officials found boxes full of uncounted paper ballots after an election was over. Reasonable accuracy and authenticity of vote counts are essential in a healthy democracy. Electronic systems have the potential for reducing some kinds of fraud and accidental loss of ballots, but they have not yet reached the level of security to ensure a reasonable degree of trust.

Stalled airports: Denver, Hong Kong, and Malaysia

In 1994, I flew over the huge Denver International Airport and the miles of wide highway leading to it. The airport covers 53 square miles, roughly twice the size of Manhattan. It was an eerie sight—nothing moved. There were no airplanes or people at the airport and no cars on the highway—ten months after the $3.2-billion airport was supposed to have opened. The opening was rescheduled at least four times. The delay cost more than $30 million per month in bond interest and operating costs. The computer-controlled baggage-handling system, which cost $193 million, caused most of the delay.[17]

The plan for the baggage system was quite ambitious. Outbound luggage checked at ticket counters or curbside counters was to travel to any part of the airport in less than ten minutes through an automated system of carts traveling at up to 19 miles per hour on 22 miles of underground tracks. Similarly, inbound luggage would go to terminals or transfer directly to connecting flights anywhere in the airport. Carts, bar-coded for their destinations, carried the bags. Laser scanners throughout the system tracked the 4,000 carts and sent information about their locations to computers. The computers used a database of flights, gates, and routing information to control motors and switches to route the carts to their destinations.

The system did not work as planned. During tests over several months, carts crashed into each other at track intersections. The system misrouted, dumped, and flung about luggage. Carts needed to move luggage went by mistake to waiting pens. Both the specific problems and the general underlying causes are instructive. Some of the specific problems were

- *Real-world problems.* Some scanners got dirty or knocked out of alignment and could not detect carts going by. Faulty latches on the carts caused luggage to fall onto the tracks between stops.

- *Problems in other systems.* The airport's electrical system could not handle the power surges associated with the baggage system. The first full-scale test blew so many circuits that the test had to be halted.

- *Software errors.* A software error caused the routing of carts to waiting pens when they were actually needed.

No one expects software and hardware of this complexity to work perfectly when first tested. In real-time systems,* especially, there are numerous interactions and conditions that designers might not anticipate. Mangling a suitcase is not embarrassing if it occurs during an early test and if the problem is fixed. It is embarrassing if it occurs after the system is in operation or if it takes a year to fix. What led to the extraordinary delay in the Denver baggage system? There seem to have been two main causes:

- *The time allowed for development and testing of the system was insufficient.* The only other baggage system of comparable size was at Frankfurt Airport in Germany. The company that built that system spent six years on development and two years testing and debugging. BAE Automated Systems, the company that built the Denver system, was asked to do it in two years. Some reports indicate that because of the electrical problems at the airport, there were only six weeks for testing.
- *Denver made significant changes in specifications after the project began.* Originally, the automated system was to serve United Airlines, but Denver officials decided to expand it to include the entire airport, making the system 14 times as large as the automated baggage system BAE had installed for United at San Francisco International Airport.

As a *PC Week* reporter said, “The bottom-line lesson is that system designers must build in plenty of test and debugging time when scaling up proven technology into a much more complicated environment.”[18] Some observers criticized BAE for taking on the job when the company should have known that there was not enough time to complete it. Others blamed the city government for poor management, politically motivated decisions, and proceeding with a grandiose but unrealistic plan.

In 2005, United Airlines scrapped the complex, trouble-plagued automated baggage system. It had cost hundreds of millions of dollars. United said it would move baggage manually.

Opening day at the new airports in Hong Kong and Kuala Lumpur were disasters. The ambitious and complex computer systems at these airports were to manage *everything*: moving 20,000 pieces of luggage per hour and coordinating and scheduling crews, gate assignments for flights, and so on. Both systems failed spectacularly. At Hong Kong’s Chek Lap Kok airport, cleaning crews and fuel trucks, baggage, passengers, and cargo went to the wrong gates, sometimes far from where their airplanes were. Airplanes scheduled to take off were empty. At Kuala Lumpur, airport employees had to write boarding passes by hand and carry luggage. Flights, of course, were delayed; food cargo rotted in the heat in Malaysia.

At both airports, the failures were blamed on people typing in incorrect information. In Hong Kong, it was perhaps a wrong gate or arrival time that was dutifully sent throughout the system. In Kuala Lumpur, mistakes by check-in agents unfamiliar with

*Real-time systems are systems that must detect and respond to or control activities of objects or people in the real world within time constraints.

the system paralyzed it. “There’s nothing wrong with the system,” said a spokesman at Malaysia’s airport. A spokesman at Hong Kong made a similar statement. They are deeply mistaken. One incorrect gate number would not have caused the problems experienced at Hong Kong. Any system that has a large number of users and a lot of user input must be designed and tested to handle input mistakes. The “system” includes more than software and hardware. It includes the people who operate it. As in the case of the Denver airport, there were questions about whether political considerations, rather than the needs of the project, determined the scheduled time for the opening of the airports.[19]

Abandoned systems

The flaws in many systems are so fundamental that the systems end up in the trash after wasting millions, or even billions, of dollars. A large British food retailer spent more than $500 million on an automated supply management system; it did not work. The Ford Motor Company abandoned a $400-million purchasing system. The California and Washington state motor vehicle departments each spent more than $40 million on computer systems before abandoning them because they never worked properly. A consortium of hotels and a rental car business spent $125 million on a comprehensive travel-industry reservation system, then canceled the project because it did not work. The state of California spent more than $100 million to develop one of the largest and most expensive state computer systems in the country: a system for tracking parents who owe child-support payments. After five years, the state abandoned the system because it did not work. It lost data, miscalculated payments, and could not communicate with other government agencies. After spending $4 billion, the IRS abandoned a tax-system modernization plan; a GAO report blamed mismanagement. The FBI spent $170 million to develop a database called the Virtual Case File system to manage evidence in investigations, then scrapped it because of many problems. A Department of Justice report blamed poorly defined and changing design requirements, lack of technical expertise, and poor management.[20] There are many more such examples.

Many projects require much more time and money than originally planned. Some are never completed because they are beyond the capabilities of the current technology. Software expert Robert Charette estimates that from 5% to 15% of information technology projects are abandoned before or soon after delivery as “hopelessly inadequate”—out of about $1 trillion spent each year worldwide. Figure 8.1 includes some reasons he cites.[21] Such large losses demand attention from computer professionals, information technology managers, business executives, and public officials who set budgets and schedules for large projects.

Legacy systems

After US Airways and America West merged, they combined their reservation systems. The self-service check-in kiosks failed. Long lines at ticket counters delayed thousands of passengers and flights. Merging different computer systems is extremely tricky, and

- ❖ Lack of clear, well-thought-out goals and specifications
- ❖ Poor management and poor communication among customers, designers, programmers, and so on
- ❖ Institutional or political pressures that encourage unrealistically low bids, unrealistically low budget requests, and underestimates of time requirements
- ❖ Use of very new technology, with unknown reliability and problems, perhaps for which software developers have insufficient experience and expertise
- ❖ Refusal to recognize or admit that a project is in trouble

Figure 8.1 Some High-level Causes of Computer-System Failures

problems are common. But this incident illustrates another factor. According to a vice president of US Airways, most airline systems date from the 1960s and 1970s. Designed for the mainframe computers of that era, they, in some cases, replaced reservations on 3×5 paper cards. These old systems "are very reliable, but very inflexible," the airline executive said.[22] These are examples of "legacy systems"—out-of-date systems (hardware, software, or peripheral equipment) still in use, often with special interfaces, conversion software, and other adaptations to make them interact with more modern systems.

The problems of legacy systems are numerous. Old hardware fails, and replacement parts are hard to find. Old software often runs on newer hardware, but it is still old software. Programmers no longer learn the old programming languages. The programmers who wrote the software or operated the systems have left the company, retired, or died. Old programs often had little or no documentation. If there were good design documents and manuals, they probably no longer exist or cannot be found. Limited computer memory led to obscure and terse programming practices. A variable a programmer might now call "flight_number" would then have been simply "f."

Why do people still use such systems? Why do you keep an old printer, computer game, or camera when you replace an old computer? Why not buy a new one? One obvious answer is cost. Another is that you might not want the overhead of learning how to use the new one. You might have lots of old documents or applications that work with the old system, and you would need to convert them. All these reasons apply, on a large scale, to legacy business systems.

Many Windows operating systems include software that mimics the older DOS operating system so that people can run old programs written for the older environment. You can find software to run old games on new computers. Modern word processors can read "ancient" document formats. And airlines integrate Web-based reservations systems and self-service check-in kiosks with very old systems.

THE Y2K PROBLEM

Many records stored on computers include a date. Many computations use dates. Older software, designed for computers with very limited storage space, typically used two digits to represent the year (e.g., 78, 95).

Many computer systems experienced problems in the 1990s when they began using dates in or after the year 2000, ("Y2K" in the jargon of the time for "Year 2000"). For example, some credit cards with expiration dates in 2000 would not work. The software interpreted the expiration date as 1900. Software to calculate payments on loans due after 2000 failed.

People writing software in the 1960s and 1970s never considered or imagined that their code would still be in use in the year 2000. But it was. It was buried in financial systems, inventory systems, medical systems, infrastructure systems, and all sorts of other systems. Businesses and governments spent many billions of dollars on the "Y2K problem," tracking down two-digit dates in their software and making modifications, rushing to finish before January 1, 2000. Programmers took special courses in old programming languages (such as COBOL, widely used for business applications) so that they could read and patch old programs. Software customers sued software vendors for the cost of searching their software for Y2K problems and fixing them.

As January 1, 2000, approached, many people feared major disasters would result as critical computer systems failed or crashed. We awoke on January 1, 2000, to find that there had been no disasters. Some systems did not work properly, but problems were not significant. Afterward, some argued that the huge multiyear, multibillion-dollar effort to upgrade systems and fix the software had succeeded. Others argued that it was an expensive waste caused by irresponsible hype. The truth includes some of both views.

What similar problems are hiding in computer systems now? What systems are not designed with a far enough look into the future?

The major users of computers in the early days included banks, airlines, government agencies, and providers of infrastructure services such as power companies. The systems grew gradually. A complete redesign and development of a fully new, modern system would, of course, be expensive. The conversion to the new system, possibly requiring some downtime, could also be very disruptive. Thus legacy systems persist.

We will continue to invent new programming languages, paradigms, and protocols—and we will later add on to the systems we develop as they age. Among the lessons legacy systems provide for computer professionals is the recognition that someone might be using your software 30 or 40 years from now. It is important to document,

document, document your work. It is important to design for flexibility, expansion, and upgrades.

8.1.4 SAFETY-CRITICAL APPLICATIONS

There are many examples of problems in safety-critical computer systems in military applications, power plants, aircraft, trains, automated factories, medical applications, and so on. We briefly look at a few aviation cases, then at one medical-instrument case in depth in the next section.

Computers in the air

The A320 Airbus airplane was the first fully "fly-by-wire" airplane. Pilots do not directly control the plane. Their actions are inputs to computers that control the aircraft systems. Between 1988 and 1993, four A320s crashed. Although investigators decided the cause for some of the crashes was "pilot error," pilots and some observers blamed the fly-by-wire system. Pilots complained that the airplane does not respond as expected, that it seems to have "a mind of its own" and may suddenly behave in unexpected and inappropriate ways. In one crash, the pilots specified a rate of descent of 3,300 feet per minute instead of the normal 800 feet per minute. The official report on the crash indicated that reasons for the error probably included the pilots' lack of familiarity with the automation equipment and confusing design of the controls and displays. The crew left the "vertical navigation" entirely to the automatic systems although there were indications that the plane was descending too fast. Perhaps they had too much confidence in the computer's ability to detect and correct mistakes. In another crash, the computer did not recognize that the plane had landed; it prevented the pilot from reversing engine thrust to brake the airplane.[23]

The Federal Aviation Administration (FAA) installed a $1-billion system at major airports for tracking high-altitude, long-distance flights. It worked well in some of the airports where it was tested. Problems occurred within a day of modifications at other airports, causing cancellation of hundreds of flights. When the FAA installed its upgraded radar system at its traffic control center in southern California, the system crashed, grounding flights nationwide. Twice, the computer that controls flights that enter British airspace broke down. Controllers used manual backup systems. The average flight delay was three hours.[24] A computer problem at the Los Angeles airport cut capacity by 50%, causing flight delays all around the country.

Air traffic control is extremely complex. The systems that manage it include computers on the ground at airports, devices in thousands of airplanes, radar, databases, communications, and so on—all of which must work in real time, tracking airplanes that move very fast. Bureaucratic and political decision-making and funding processes compound the inherent difficulty of the task.

We discuss more incidents of failures of aircraft systems in Sections 8.3 and 8.4, where we look at solutions (including better computer systems).

8.1.5 PERSPECTIVES ON FAILURE

How close to perfection should we expect billing systems to be? A water-utility company sent a customer an incorrect bill for $22,000. A spokesman for the company pointed out that one incorrect bill out of 275,000 monthly bills is a good error rate. Is that reasonable? How accurate should the software for ATMs be? The double-withdrawal incident mentioned in Chapter 1 affected roughly 150,000 transactions. With approximately eight billion ATM transactions each year, that was one error in roughly 45,000. Is that an acceptable rate? (There were probably other ATM errors in that year, but the publicity given to this case suggests that it affected far more transactions than other errors.) How accurate should software for check processing be? 99%? 99.9%? U.S. financial institutions process roughly 250 million checks per day. Even if they made errors on 10,000 checks every day, the accuracy rate would be better than 99.99%. (How high would the accuracy rate be if checks and bills were processed without computers?) At some point, the expense of improving a system is not worth the gain, especially for applications where errors can be detected and corrected at lower cost than it would take to try to eliminate them.

Many large, complex, expensive computer systems work extremely well. We rely on them daily. Many computer systems protect lives and increase safety. To balance the examples we described of failures, we mention a few examples of successes.

The New York Stock Exchange prepared well for spikes in trading. On one day, the Stock Exchange computers processed 76% more trades than the previous record. They handled the sales without errors or delays. The Exchange managers had planned in advance, spending $2 billion on a system with hundreds of computers, 200 miles of fiber-optic cable, 8,000 telephone circuits, and 300 data routers. They had spent weekends testing the system on triple and quadruple the normal trading volume. Similarly, over one weekend, Barclays Bank replaced three incompatible computer systems with a new system to handle 25 million accounts. The transition, estimated to cost more than £100 million, was successful.[25]

A ground-proximity warning system (GPWS) helps prevent airplanes from crashing into mountains (a major cause of air travel fatalities). Older radar-based systems sometimes gave warning only ten seconds before a potential impact. The GPWS contains a digital map of the world's topography. It can give a pilot up to a minute of warning if a plane is too close to a mountain and automatically displays a map of nearby mountains. Dangerous peaks are shown in red. The system works where the older system did not: The radar-based system could not distinguish between approaching the ground for landing and a potential crash, so it did not give warnings during a landing approach. The digital maps enable the computer system to make the distinction. The GWPS is likely responsible for preventing crashes in eight incidents since 2000 (in which pilots incorrectly set an altimeter, attempted to land with poor visibility, mistook building lights for airport lights, and so on). In some airplanes, when visibility is poor, a computer system displays an accurate view of the terrain on the plane's windshield. Pilots see a view close to what they would see in clear weather.[26]

Overall, computers and other technologies have made air travel safer. In the middle of the first decade of this century, there was roughly one fatal accident per four million commercial flights, down 60% from ten years earlier.[27]

Trust the human or the computer system?

How much control should computers have in a crisis? This question arises in many application areas. We address it in the context of aircraft systems.

The Traffic Collision Avoidance System (TCAS) detects a potential in-air collision of two airplanes and directs the pilots to avoid each other. The first version of the system had so many false alarms that it was unusable. In some incidents, the system directed pilots to fly toward each other rather than away, potentially causing a collision instead of avoiding one. TCAS was improved, however. It is a great advance in safety, according to the head of the Airline Pilots Association's safety committee.[28] TCAS functioned correctly on a Russian airplane carrying many children and on a German cargo plane. The systems detected a potential collision and told the Russian pilot to climb and the German pilot to descend. Unfortunately, the Russian pilot followed an air traffic controller's instruction to descend, and the planes collided. In this example, the computer's instructions were better than the human's. A few months after this tragedy, the pilot of a Lufthansa 747 ignored instructions from an air traffic controller and followed instructions from the computer system instead, avoiding a midair collision. U.S. and European pilots are now trained to follow TCAS instructions even if they conflict with instructions from an air traffic controller.

Pilots are trained to immediately turn off autopilot systems when TCAS signals a potential collision. They manually maneuver the plane to avoid the collision. That might change. The Airbus 380, the world's largest passenger airplane, began flying in 2006. Its pilots are trained to allow its autopilot system to control the plane when a midair collision threatens. The aircraft maker says that pilots sometimes overreact to collision warnings and make extreme maneuvers that can injure passengers or cause a collision with other air traffic in the area. The policy is controversial among pilots.[29]

Some airlines (and private pilots) disable parts of computer systems in their planes because they do not trust the computer or because they believe the pilot should more actively fly the plane. However, there are circumstances where the computer can do better than most people. Like anti-lock braking systems in automobiles that control braking to avoid skidding, computer systems in airplanes control sudden sharp climbs to avoid stalling. The computers in some airplanes prevent certain actions even if the pilot tries them (e.g., banking at a very steep angle). Some people object, arguing that the pilot should have ultimate control in case unusual action is needed in an emergency. Based on accident statistics, some airlines believe that preventing pilots from doing something "stupid" can save more lives than letting them do something bold and heroic, but outside the program limitations, in the very rare cases where it might be necessary.

8.2 Case Study: The Therac-25

8.2.1 THERAC-25 RADIATION OVERDOSES

The benefits of computing technology to health care are numerous and very impressive. They include improved diagnosis, monitoring of health conditions, development of new drugs, information systems that speed treatment and reduce errors, devices that save lives, devices that increase safety of surgeries, and on and on. Yet one of the classic case studies of a deadly software failure is a medical device: a radiation treatment machine.

The Therac-25 was a software-controlled radiation-therapy machine used to treat people with cancer. Between 1985 and 1987, Therac-25 machines at four medical centers gave massive overdoses of radiation to six patients. In some cases, the operator repeated an overdose because the machine's display indicated that it had given no dose. Medical personnel later estimated that some patients received between 13,000 and 25,000 rads,* where the intended dose was in the 100–200 rad range. These incidents caused severe and painful injuries and the deaths of three patients. Why is it important to study a case as old as this? To avoid repeating the errors. Medical physicists operating a different radiation-treatment machine in Panama in 2000 tried to circumvent a limitation in the software in an attempt to provide more shielding for patients. Their actions caused dosage miscalculations. Twenty-eight patients received overdoses of radiation, and several died.[30] It seems that dramatic lessons need repetition with each new generation.

What went wrong with the Therac-25?

Studies of the Therac-25 incidents showed that many factors contributed to the injuries and deaths. The factors include lapses in good safety design, insufficient testing, bugs in the software that controlled the machines, and an inadequate system of reporting and investigating the accidents. (Articles by computer scientists Nancy Leveson and Clark Turner and by Jonathan Jacky are the main sources for this discussion.[31])

To understand the discussion of the problems, it will help to know a little about the machine. The Therac-25 is a dual-mode machine. That is, it can generate an electron beam or an X-ray photon beam. The type of beam needed depends on the tumor being treated. The machine's linear accelerator produces a high-energy electron beam (25 million electron volts) that is dangerous. Patients must not be exposed to the raw beam. A computer monitors and controls movement of a turntable that holds three sets of devices. Depending on whether the treatment is electron or X-ray, the machine rotates a different set of devices in front of the beam to spread it and make it safe. It is essential that the proper protective device be in place when the electron beam is on. A third position of the turntable uses a light beam instead of the electron beam to help the operator position the beam precisely in the correct place on the patient's body.

*A rad is the unit used to quantify radiation doses. It stands for "radiation absorbed dose."

8.2.2 SOFTWARE AND DESIGN PROBLEMS

Design flaws

The Therac-25 followed earlier machines called the Therac-6 and Therac-20. It differed from them in that it was fully computer controlled. The older machines had hardware safety interlock mechanisms, independent of the computer, that prevented the beam from firing in unsafe conditions. The design of the Therac-25 eliminated many of these hardware safety features. The Therac-25 reused some software from the Therac-20 and Therac-6. The software was apparently assumed to be functioning correctly. This assumption was wrong. When new operators used the Therac-20 there were frequent shutdowns and blown fuses, but no overdoses. The Therac-20 software had bugs, but the hardware safety mechanisms were doing their job. Either the manufacturers did not know of the problems with the Therac-20 or they completely missed the serious implications.

The Therac-25 malfunctioned frequently. One facility said there were sometimes 40 dose rate malfunctions in a day, generally underdoses. Thus, operators became used to error messages appearing often, with no indication that there might be safety hazards.

There were a number of weaknesses in the design of the operator interface. The error messages that appeared on the display were simply error numbers or obscure messages ("Malfunction 54" or "H-tilt"). This was not unusual for early computer programs when computers had much less memory and mass storage than they have now. One had to look up each error number in a manual for more explanation. The operator's manual for the Therac-25, however, did not include any explanation of the error messages. The maintenance manual did not explain them either. The machine distinguished between errors by the amount of effort needed to continue operation. For certain error conditions, the machine paused, and the operator could proceed (turn on the electron beam) by pressing one key. For other kinds of errors, the machine suspended operation and had to be completely reset. One would presume that the machine would allow one-key resumption only after minor, not safety-related, errors. Yet one-key resumption occurred in some of the accidents in which patients received multiple overdoses.

Atomic Energy of Canada Limited (AECL), a Canadian government corporation, manufactured the Therac-25. Investigators studying the accidents found that AECL produced very little documentation concerning the software specifications or the testing plan during development of the program. Although AECL claimed that they tested the machine extensively, it appeared that the test plan was inadequate.

Bugs

Investigators were able to trace some of the overdoses to two specific software errors. Because many readers of this book are computer science students, I will describe the bugs. These descriptions illustrate the importance of using good programming techniques. However, some readers have little or no programming knowledge, so I will simplify the descriptions.

After the operator entered treatment parameters at a control console, a software procedure called Set-Up Test performed a variety of checks to be sure the machine was in the correct position, and so on. If anything was not ready, this procedure scheduled itself to rerun the checks. (The system might simply have to wait for the turntable to move into place.) The Set-Up Test procedure can run several hundred times while setting up for one treatment. A flag variable indicated whether a specific device on the machine was in the correct position. A zero value meant the device was ready; a nonzero value meant it must be checked. To ensure that the device was checked, each time the Set-Up Test procedure ran, it incremented the variable to make it nonzero. The problem was that the flag variable was stored in one byte. After the 256th call to the routine, the flag overflowed and showed a value of zero. (If you are not familiar with programming, think of this as an automobile's odometer rolling over to zero after reaching the highest number it can show.) If everything else happened to be ready at that point, the program did not check the device position, and the treatment could proceed. Investigators believe that in some of the accidents, this bug allowed the electron beam to be on when the turntable was positioned for use of the light beam, and there was no protective device in place to attenuate the beam.

Part of the tragedy in this case is that the error was such a simple one, with a simple correction. No good student programmer should have made this error. The solution is to set the flag variable to a fixed value, say 1, rather than incrementing it, to indicate that the device needs checking.

Other bugs caused the machine to ignore changes or corrections made by the operator at the console. When the operator typed in all the necessary information for a treatment, the program began moving various devices into place. This process could take several seconds. The software checked for editing of the input by the operator during this time and restarted the setup if it detected editing. However, because of bugs in this section of the program, some parts of the program learned of the edited information while others did not. This led to machine settings that were incorrect and inconsistent with safe treatment. According to the later investigation by the Food and Drug Administration (FDA), there appeared to be no consistency checks in the program. The error was most likely to occur with an experienced operator who was quick at editing input.

In a real-time, multitasking system that controls physical machinery while an operator enters—and may modify—input, there are many complex factors that can contribute to subtle, intermittent, and hard-to-detect bugs. Programmers working on such systems must learn to be aware of the potential problems and to use good programming practices to avoid them.

8.2.3 WHY SO MANY INCIDENTS?

There were six known Therac-25 overdoses. You may wonder why hospitals and clinics continued to use the machine after the first one.

The Therac-25 had been in service for up to two years at some clinics. They did not immediately pull it from service after the first few accidents because they did not know immediately that it caused the injuries. Medical staff members considered various other explanations. The staff at the site of the first incident said that one reason they were not certain of the source of the patient's injuries was that they had never seen such a massive radiation overdose before. They questioned the manufacturer about the possibility of overdoses, but the company responded (after the first, third, and fourth accidents) that the machine could not have caused the patient injuries. According to the Leveson and Turner investigative report, they also told the facilities that there had been no similar cases of injuries.

After the second accident, AECL investigated and found several problems related to the turntable (not including any of the ones we described). They made some changes in the system and recommended operational changes. They declared that they had improved the safety of the machine by five orders of magnitude, although they told the FDA that they were not certain of the exact cause of the accident. That is, they did not know whether they had found the problem that caused the accident or just other problems. In making decisions about continued use of the machines, the hospitals and clinics had to consider the costs of removing the expensive machine from service (in lost income and loss of treatment for patients who needed it), the uncertainty about whether the machine was the cause of the injuries, and, later, when that was clear, the manufacturer's assurances that they had solved the problem.

A canadian government agency and some hospitals using the Therac-25 made recommendations for many more changes to enhance safety; they were not implemented. After the fifth accident, the FDA declared the machine defective and ordered AECL to inform users of the problems. The FDA and AECL spent about a year (during which the sixth accident occurred) negotiating about changes in the machine. The final plan included more than two dozen changes. They eventually installed the critical hardware safety interlocks, and most of the machines remained in use with no new incidents of overdoses after 1987.[32]

Overconfidence

In the first overdose incident, when the patient told the machine operator that the machine had "burned" her, the operator told her that was impossible. This was one of many indications that the makers and some users of the Therac-25 were overconfident about the safety of the system. The most obvious and critical indication of overconfidence in the software was the decision to eliminate the hardware safety mechanisms. A safety analysis of the machine done by AECL years before the accidents suggests that they did not expect significant problems from software errors. In one case where a clinic added its own hardware safety features to the machine, AECL told them it was not necessary. (None of the accidents occurred at that facility.)

The hospitals using the machine assumed that it worked safely, an understandable assumption. Some of their actions, though, suggest overconfidence or at least practices that they should have avoided. For example, operators ignored error messages because the machine produced so many of them. A camera in the treatment room and an intercom system enabled the operator to monitor the treatment and communicate with the patient. (The operator uses a console outside the shielded treatment room.) On the day of an accident at one facility, neither the video monitor nor the intercom was functioning. The operator did not see or hear the patient try to get up after an overdose. He received a second overdose before he reached the door and pounded on it. This facility had successfully treated more than 500 patients with the machine before this incident.

8.2.4 OBSERVATIONS AND PERSPECTIVE

From design decisions all the way to responding to the overdose accidents, the manufacturer of the Therac-25 did a poor job. Minor design and implementation errors usually occur in a complex system, but the number and pattern of problems in this case, and the way they were handled, suggest serious irresponsibility. This case illustrates many of the things that a responsible, ethical software developer should not do. It illustrates the importance of following good procedures in software development. It is a stark reminder of the consequences of carelessness, cutting corners, unprofessional work, and attempts to avoid responsibility. It reminds us that a complex system can work correctly hundreds of times with a bug that shows up only in unusual circumstances—hence the importance of always following good safety procedures in operation of potentially dangerous equipment. This case also illustrates the importance of individual initiative and responsibility. Recall that some facilities installed hardware safety devices on their Therac-25 machines. They recognized the risks and took action to reduce them. The hospital physicist at one of the facilities where the Therac-25 overdosed patients spent many hours working with the machine to try to reproduce the conditions under which the overdoses occurred. With little support or information from the manufacturer, he was able to figure out the cause of some of the malfunctions.

To put the Therac-25 in perspective, it is helpful to remember that failures and other accidents have always occurred and continue to occur in systems that do not use computers. Two other linear accelerator radiation-treatment machines seriously overdosed patients. Three patients received overdoses in one day at a London hospital in 1966 when safety controls failed. Twenty-four patients received overdoses from a malfunctioning machine at a Spanish hospital in 1991; three patients died. Neither of these machines had computer controls.[33]

Two news reporters reviewed more than 4,000 cases of radiation overdoses reported to the U.S. government. Here are a few of the overdose incidents they describe. A technician started a treatment, then left the patient for 10–15 minutes to attend an office party. A technician failed to carefully check the prescribed treatment time. A technician failed to measure the radioactive drugs administered; she just used what looked like the right amount. In at least two cases, technicians confused microcuries and millicuries.*

*A curie is a measure of radioactivity. A millicurie is one thousand times as much as a microcurie.

The underlying problems were carelessness, lack of appreciation for the risk involved, poor training, and lack of sufficient penalty to encourage better practices. In most cases, the medical facilities paid small fines or none at all.[34]

Most of the incidents we just described occurred in systems without computers. For some, a good computer system might have prevented the problem. Many could have occurred whether or not the treatment system was controlled by a computer. These examples do not excuse the Therac-25. They suggest, however, that individual and management responsibility, good training, and accountability are important no matter what technology we use.

8.3 Increasing Reliability and Safety

Success actually requires avoiding many separate possible causes of failure.

—Jared Diamond[35]

8.3.1 WHAT GOES WRONG?

Computer systems fail for two general reasons: The job they are doing is inherently difficult, and the job is often done poorly. Several factors combine to make the task difficult. Early computer programs were fed some numbers, did mathematical computations, and provided some answers. There were sometimes errors, but the task was not extremely complex. Computer systems now interact with the real world (including both machinery and unpredictable humans), include complex communications networks, have numerous features and interconnected subsystems, and are extremely large. A cell phone has several millions of lines of computer code. General Motors estimated that its cars would have 100 million lines of code by 2010.[36] Computer software is "nonlinear" in the sense that, whereas a small error in an engineering project might cause a small degradation in performance, a single typo in a computer program can cause a dramatic difference in behavior.

The job can be done poorly at any of many stages, from system design and implementation to system management and use. (This characteristic is not unique to computer systems, of course. We can say the same about building a bridge, a space shuttle, a car, or any complex system in the modern world.) Figure 8.1 (in Section 8.1.3) summarized high-level, management-related causes of system failures. Figure 8.2 lists more factors in computer errors and system failures. The examples we described illustrate most of them. We comment on a few.

Overconfidence

Overconfidence, or an unrealistic or inadequate understanding of the risks in a complex computer system, is a core issue. When system developers and users appreciate the risks,

- ❖ Design and development
  - Inadequate attention to potential safety risks.
  - Interaction with physical devices that do not work as expected.
  - Incompatibility of software and hardware or of application software and the operating system.
  - Not planning and designing for unexpected inputs or circumstances.
  - Insufficient testing.
  - Reuse of software from another system without adequate checking.
  - Overconfidence in software.
  - Carelessness.
- ❖ Management and use
  - Data-entry errors.
  - Inadequate training of users.
  - Errors in interpreting results or output.
  - Failure to keep information in databases up to date.
  - Overconfidence in software by users.
  - Insufficient planning for failures, no backup systems or procedures.
- ❖ Misrepresentation, hiding problems, and inadequate response to reported problems.
- ❖ Insufficient market or legal incentives to do a better job.

Figure 8.2 Some Factors in Computer-System Errors and Failures

they have more motivation to use the techniques that are available to build more reliable and safer systems and to be responsible users. How many PC users never back up their files until after a disk crash that destroys critical data or months of work?

Some safety-critical systems that failed had supposedly "fail-safe" computer controls. In some cases, the logic of the program was fine, but the failure resulted from not considering how the system interacts with real users or real-world problems (such as loose wires or fallen leaves on train tracks).

Can we analyze and quantify the risks of failure in a system? Yes, but we must use the techniques for developing estimates of failure rates carefully. For example, the computers on the A320 airplane each have redundant software systems designed by two separate teams of programmers. The redundancy is a safety feature, but how much safety does it provide? The failure rate was supposed to be less than one failure per billion flight hours. Designers calculated it by multiplying the estimated failure rates of each of the two systems, one in 100,000 hours. The calculation is reasonable if the systems are entirely independent. But safety experts say that even when programmers work separately, they tend to make the same kinds of errors, especially if there is an error, ambiguity, or omission in the program specifications.[37]

Unrealistic reliability or safety estimates can come from genuine lack of understanding, from carelessness, or from intentional misrepresentation. People without a high regard for honesty sometimes give in to business or political pressure to exaggerate safety, to hide flaws, to avoid unfavorable publicity, or to avoid the expense of corrections or lawsuits.

Reuse of software: the Ariane 5 rocket and "No Fly" lists

Less than 40 seconds after the launch of the first Ariane 5 rocket, the rocket veered off course and was destroyed as a safety precaution. The rocket and the satellites it was carrying cost approximately $500 million. A software error caused the failure.[38] The Ariane 5 used some software designed for the earlier, successful Ariane 4. The software included a module that ran for about a minute after initiation of a launch on the Ariane 4. It did not have to run after takeoff of the Ariane 5, but a decision was made to avoid introducing new errors by making changes in a module that operated well in Ariane 4. This module did calculations related to velocity. The Ariane 5 travels faster than the Ariane 4 after takeoff. The calculations produced numbers bigger than the program could handle (an "overflow" in technical jargon), causing the system to halt.

A woman named Jan Adams, and many other people with first initial J and last name Adams, were flagged as possible terrorists when they tried to board an airplane. The name "Joseph Adams" is on a "No Fly" list of suspected terrorists (and other people considered safety threats) given to the airlines by the Transportation Security Agency. To compare passenger names with those on the "No Fly" list, some airlines used old software and strategies designed to help ticket agents quickly locate a passenger's reservation record (e.g., if the passenger calls in with a question or to make a change). The software searches quickly and "casts a wide net." That is, it finds any possible match, which a sales agent can then verify. In the intended applications for the software, there is no inconvenience to anyone if the program presents the agent with a few potential matches of similar names. In the context of tagging people as possible terrorists, a person mistakenly "matched" will likely undergo questioning and extra luggage and body searches by security agents.

Does something about these incidents sound familiar? The Therac-25 (Section 8.2) used software from earlier machines, the Therac-6 and Therac-20. The software seemed to function acceptably in those systems but was not appropriate or safe for the Therac-25. Should we not reuse software? One of the goals of programming paradigms like object-oriented code is to make software elements that can be widely used, thus saving time and effort. Reuse of working software should also increase safety and reliability. After all, it has undergone field testing in a real, operational environment; we know it works. At least, we think it works. The critical point is that it works in a different environment. It is essential to reexamine the specifications and design of the software, consider implications and risks for the new environment, and retest the software for the new use. U.S. and

European aircraft safety experts now apply this cautious policy. They recommend that when an aircraft system uses commercial, off-the-shelf software, it be required to meet the same extensive review and testing as new custom software for aircraft. When a dispute arose about off-the-shelf software in the development of the Airbus 380 airplane, the dispute was about the level and quality of the required certification, not over whether it was a necessary step.

Failure to update

Failure to update information in databases has led to problems of varying severity. When considering the next two examples and other possible examples, think about what kinds of databases are inherently difficult to keep up-to-date. Which are extremely expensive, possibly too expensive, to keep up-to-date? For which is updating very important? For which, not so important?

The FBI maintains a database that contains criminal history records from participating states and is available to police throughout most of the country. *Privacy Journal* reported that about half the records do not indicate whether a suspect was convicted or exonerated. (In addition, there was a high inaccuracy rate.)[39]

The U.S. government spent $900 million to set up a database to track the roughly two million foreign visitors who arrive in the U.S. each month. Goals include screening for terrorists and detecting visitors who stay longer than legally permitted. The system stores photos and fingerprints of foreigners but does not have a way for visitors to check out when leaving the country. Several airports are testing a check-out system, but a full system at all border exits is likely infeasible. Without a way to update records when a visitor leaves, the system cannot meet one of its main goals.[40]

What design features in databases can encourage updating or reduce problems resulting from out-of-date information?

8.3.2 PROFESSIONAL TECHNIQUES

Software engineering and professional responsibility

The many examples of computer system errors and failures suggest the importance of using good software engineering techniques at all stages of development, including specifications, design, implementation, documentation, and testing. There is a wide range between poor work and good work, as there is in virtually any field. Professionals, both programmers and managers, have the responsibility to study and use the techniques and tools that are available and to follow the procedures and guidelines established in the various relevant codes of ethics and professional practices. (The ACM/IEEE-CS Software Engineering Code of Ethics and Professional Practice and the ACM Code of Ethics and Professional Conduct, in Appendix A, are two important sets of general guidelines for such practices.) A subfield of computer science focusing on design and development of safety-critical software is growing. Safety specialists emphasize that developers must "design in" safety from the start. There are techniques of hazard analysis that help system designers identify risks and protect against them. Software engineers who work on safety-critical applications should have special training. Software expert Nancy Leveson emphasizes that with good technical practices and good management, you can develop large systems right: "One lesson is that most accidents are not the result of unknown scientific principles but rather of a failure to apply well-known, standard engineering practices. A second lesson is that accidents will not be prevented by technological fixes alone, but will require control of all aspects of the development and operation of the system."[41]

To complete a large project successfully requires long and careful planning and good management. Companies that do well expend extensive effort to learn the needs of the client and to understand how the client will use the system. Good software developers help clients better understand their own goals and requirements, which the clients might not be good at articulating. Often some aspects of the way they work or what they need and expect are so obvious to them that they do not state them to the system developers. The long planning stage allows for discovering and modifying unrealistic goals. One company that developed a successful financial system that processes one trillion dollars in transactions per day spent several years developing specifications for the system, then only six months programming, followed by carefully designed, extensive testing.

Mistakes will happen, and unexpected problems will arise. An atmosphere of open, honest communication within the software development company and between the company and the client is essential for learning of problems early and minimizing the effort required to handle them.

In the rest of this section, we look at a few specific aspects of designing and developing good software.

User interfaces and human factors

If you are using a word processor to edit a document and you try to quit without saving your changes, what happens? Most programs will remind you that you have not saved your changes and give you a chance to do so. The designers of the programs know that people forget or sometimes click or type the wrong command. This is a simple and common example of considering human factors in designing software—one that has avoided personal calamities for millions of people.

Well-designed user interfaces can help avoid many computer-related problems. System designers and programmers need to learn from psychologists and human-factor experts who know principles and practices for doing a good job.* User interfaces should provide clear instructions and error messages. They should be consistent. They should include appropriate checking of input to reduce major system failures caused by typos or other errors a person will likely make.

The crash of American Airlines Flight 965 near Cali, Columbia, in 1995 illustrates the importance of consistency (and other aspects of good user interfaces). While approaching

*See, for example, the Shneiderman, Tufte, Nielsen, and Norman books in the list of references at the end of the chapter.

the airport, the pilot intended to lock the autopilot onto the beacon, called Rozo, that would lead the plane to the airport. The pilot typed "R," and the computer system displayed six beacons beginning with "R." Normally, the closest beacon is at the top of the list. The pilot selected it without checking carefully. The beacon at the top of the list was "Romeo" and was more than 100 miles away, near Bogota. The plane turned more than 90 degrees and headed for Romeo. In the dark, it crashed into a mountain, killing 159 people.[42]

In the lawsuits that followed, juries attributed blame mostly to pilot error. The pilot chose the wrong beacon without checking and continued to descend at night after the plane made a large, unexpected turn. A jury assigned some of the responsibility to the companies that provided the computer system. While it is clear that the pilot could have and should have avoided the crash, it is also clear that the inconsistency in the display—not putting the nearest beacon at the top of the list—created a dangerous situation.

As an illustration of more principles that can help build safer systems, we consider other aspects of automated flight systems. An expert in this area emphasizes the following points:[43]

- *The pilot needs feedback to understand what the automated system is doing at any time.* This is critical when the pilot must suddenly take over if the automation fails or if he or she must turn it off for any reason. One example is having the throttle move as a manually operated throttle would, even though movement is not necessary when the automated system is operating.
- *The system should behave as the pilot (or, in general, experienced user) expects.* Pilots tend to reduce their rate of climb as they get close to their desired altitude. On the McDonnell Douglas MD-80, the automated system maintains a climb rate that is up to eight times as fast as pilots typically choose. Pilots, concerned that the plane might overshoot its target altitude, made adjustments, not realizing that their intervention turned off the automated function that caused the plane to level out when it reached the desired altitude. Thus, because the automation behaved in an unexpected way, the airplane climbed too high—exactly what the pilot was trying to prevent. (The incidence of the problem declined with more training.)
- *A workload that is too low can be dangerous.* Clearly, an overworked operator is more likely to make mistakes. One of the goals of automation is to reduce the human workload. However, a workload that is too low can lead to boredom, inattention, or lack of awareness of the current status. That is a danger if the pilot must take over in a hurry.

Good user interfaces are essential in safety-critical applications. They are important also in ordinary applications. Customers do not return to Web sites that are confusing and difficult to navigate. It is not a coincidence that some of the most popular Web sites are, according to design experts, some of the best designed.

Redundancy and self-checking

Redundancy and self-checking are two techniques important in systems on which lives and fortunes depend. The space shuttles in the 1980s used four identical but independent computer systems that received input from multiple sensors and checked their results against each other. If one computer disagreed with the other three, it was taken out of service. If two of the three remaining judged the third to be faulty, it was taken out of service, and the rest of the flight canceled. In case of a more serious problem, perhaps caused by a common flaw in the software, there was a fifth computer, made by another manufacturer and programmed by different programmers, that could control the descent of the shuttle.[44] This degree of redundancy is expensive and there are not many applications that use it. It illustrates the kinds of precautions that system designers can take for systems that operate in dangerous physical environments where human lives are at stake or in other systems where a failure would have disastrous effects.

Complex systems collect information on their own activity for use in diagnosing and correcting errors. AT&T's telephone system handles roughly 100 million calls a day. The system constantly monitors itself and corrects problems automatically. Half of the computing power of the system goes to checking the rest for errors. When it detects a problem in a switching component, the component automatically suspends use of the switch, informs the rest of the network that it is out of service temporarily and should not receive calls, activates recovery routines that take a few seconds to correct the problem, then informs the network that the component is functioning again. But wait a minute! This is the same system that failed several years ago, disrupting telephone service for hours. In fact, it was this very part of the system that caused the breakdown. There was a bug in the routine that processed recovery messages from switches that had failed and recovered. The same software operated in each switch. Thus, each switch that received the message failed, then recovered and sent a recovery message. A chain reaction of failures occurred. The bug was in a software upgrade that had been running for about a month.[45] Even when we follow the best professional practices, with extensive testing, we have no guarantee that such complex systems are bug free.

Testing

It is difficult to overemphasize the importance of adequate, well-planned testing of software. Testing is not arbitrary. There are principles and techniques for doing a good job. Many significant computer system failures in previously working systems occurred soon after installation of an update or upgrade. Even small changes need thorough testing. Unfortunately, many cost-conscious managers, programmers, and software developers see testing as a dispensable luxury, a step you can skimp on to meet a deadline or to save money. This is a common, but foolish, risky, and often irresponsible attitude.

In his investigation of the destruction of the Challenger space shuttle, Richard Feynman concluded that development of the onboard computer systems for the space shuttle used good safety criteria and testing plans.* Ironically, he was told that, because the shuttle software usually passed its tests, NASA management planned to reduce testing to save money. Fortunately, instead, as a result of studies after the loss of the Challenger, NASA instituted a practice called independent verification and validation (IV&V). This means that an independent company (i.e., not the one that developed the program and not the customer) tests and validates the software. (Testing and verification by an independent organization is not practical for all projects, but many software developers have their own testing teams that are independent of the programmers who develop a system.) The IV&V team acts as "adversaries" and tries to find flaws. A few years later, NASA planned to eliminate IV&V but switched direction again. In response to several studies, including an extensive one done by software safety experts, NASA decided to make IV&V a permanent part of the program.[46] This example illustrates a common ambivalence about testing.

You might have used a *beta version* of a product or heard of *beta testing*. Beta testing is a near-final stage of testing. A selected set of customers (or members of the public) use a complete, presumably well-tested system in their "real-world" environment. Thus, this is testing by regular users, not software experts. Beta testing can detect software limitations and bugs that the designers, programmers, and testers missed. It can also uncover confusing aspects of user interfaces, need for more rugged hardware, problems that occur when interfacing with other systems or when running a new program on older computers, and many other sorts of problems.

8.3.3 LAW, REGULATION, AND MARKETS

Criminal and civil penalties

Legal remedies for faulty systems include suits against the company that developed or sold the system and criminal charges when fraud or criminal negligence occurs. Families of Therac-25 victims sued; they settled out of court. A bank won a large judgment against a software company for a faulty financial system that caused problems a user described as "catastrophic."[47] Several people have won large judgments against credit bureaus for incorrect data in credit reports that caused havoc in their lives.

Many contracts for business computer systems limit the amount the customer can recover to the actual amount spent on the computer system. Customers know, when they sign the contract, that there is generally no coverage for losses incurred because the system did not meet their needs for any reason. Courts have upheld such contract limitations. If people and businesses cannot count on the legal system upholding the terms of a contract, contracts would be almost useless. Millions of business interactions that take

*Seals that failed in cold weather caused the destruction of the Challenger, not a software error. Experts afterward studied many aspects of safety.

place daily would become more risky and therefore more expensive. Because fraud and misrepresentation are not, of course, part of a contract, some companies that suffer large losses allege fraud and misrepresentation by the seller in an attempt to recover some of the losses, regardless of whether the allegations have firm grounding.

Well-designed liability laws and criminal laws—not so extreme that they discourage innovation but clear and strong enough to provide incentives to produce good systems—are important legal tools for increasing reliability and safety of computer systems, as they are for protecting privacy and for protecting customers in other industries. After-the-fact penalties do not undo the injuries that occurred, but the prospect of paying for mistakes and sloppiness is incentive to be responsible and careful. Payments compensate the victim and provide some justice. An individual, business, or government that does not have to pay for its mistakes and irresponsible actions will make more of them. These observations about liability apply as well to accuracy of private (business) databases. The lack of market incentives makes achieving and maintaining accuracy in government databases more difficult. In addition, you often cannot sue the government. Outside of government, we are more likely to pay for accidents and carelessness.

Unfortunately, there are many flaws in liability law in the United States. People often win multimillion-dollar suits when there is no scientific evidence or sensible reason to hold the manufacturer or seller of a product responsible for accidents or other negative impacts. Abuse of the liability lawsuit system almost shut down the small-airplane manufacturing industry in the U.S. for years. The complexity of large computer systems make designing liability standards difficult, but this is a necessary task.

Warranties for consumer software

Most mass, retail consumer software, from photo-management programs to games, comes packaged with "shrink-wrap" or "click-on" licensing agreements that indicate you are buying the software "as-is." There is no guarantee that it works correctly. Some such agreements also include provisions prohibiting the user from publically criticizing the software. Some include provisions that the vendor may choose the state in which legal disputes are settled, possibly at great inconvenience to the consumer. Consumer advocates and the software industry disagree on the extent to which the law should uphold these agreements. At one extreme, the agreements would be binding contracts. At the other extreme, some consumer advocates argue for mandatory warranties on software, for making software companies pay for bugs, and for voiding provisions of licensing agreements they consider unfair to consumers.

Supporters of strict legal requirements for warranties argue that such requirements would encourage responsibility on the part of software sellers and produce better software. Consumers would have legal protection against large, indifferent companies. Consumers are paying for a product that works, and fairness dictates that they get one that does.

Opponents point out that such requirements would raise prices. The additional cost of development, testing, and insurance would hurt small companies most, putting them out of business and leading to concentration of the industry in large companies. The costs would also reduce innovation and development of new software. The inherent complexity of software makes production of error-free software infeasible. Actual use by consumers is an important part of the testing that leads to corrections and upgrades. (Some companies pay users a few dollars for each bug they find.)

Supporters of stricter laws might respond that we would not accept these arguments if we were discussing, say, microwave ovens. If we buy one, we expect it to work. A basic issue in this debate is whether legal standards and requirements for quality and warranties should be the same for software as for physical products. Some related questions are the following: How well did the first microwave ovens, the first cars, the first record players work? Are we in the early stages of software as a product? Are consumers willing to accept some bugs in software as a trade-off for other benefits? Would a warranty law speed up the process of improving software?

Having different liability standards for software and physical products raises some problems. A microwave oven has embedded software. Can the sellers of microwave ovens claim that the weaker standards for software should cover their product? Would companies making various devices and appliances add software to their products unnecessarily, just to bring the products under the lower software standards? Would this lead to a serious decline in product quality? Although there are good arguments for treating consumer software differently from physical products, laws would have to define carefully the category of products to which they apply.[48]

Regulation and safety-critical applications

Is there legislation or regulation that can prevent life-threatening computer failures? A law saying that a radiation machine should not overdose a patient would be silly. We know that it should not do that. We could ban the use of computer control for applications where an error could be fatal, but such a ban is ill advised. In many applications, the benefits of using computers are well worth the risks.

A widely accepted option is regulation, possibly including specific testing requirements and requirement for approval by a government agency before a new product can be sold. The FDA has regulated drugs and medical devices for decades. Companies must do extensive testing, provide huge quantities of documentation, and get government approval before they sell new drugs and some medical devices. Arguments in favor of such regulation, both for drugs and for safety-critical computer systems include the following: Most potential customers and people who would be at risk (e.g., patients) do not have the expertise to judge the safety or reliability of a system. It is better to prevent use of a bad product than to rely on after-the-calamity remedies. It is too difficult and expensive for ordinary people to sue large companies successfully.

If the FDA had thoroughly examined the Therac-25 before it was put into operation, it might have found the flaws before any patients were injured. However, we should note some weaknesses and trade-offs in the regulatory approach.[49] The approval process is extremely expensive and time-consuming. The multiyear delays in introducing a good product cost many lives. Political concerns affect the approval process. Competitors influence decisions. Also, there is an incentive for bureaucrats and regulators to be overcautious. Damage caused by an approved product results in bad publicity and possible firing for the regulator who approved it. Deaths or losses caused by the delay or failure to approve a good new product are usually not obvious and get little publicity.

Leveson and Turner, in their Therac-25 article, summarize some of these dilemmas:

> The issues involved in regulation of risky technology are complex. Overly strict standards can inhibit progress, require techniques behind the state of the art, and transfer responsibility from the manufacturer to the government. The fixing of responsibility requires a delicate balance. Someone must represent the public's needs, which may be subsumed by a company's desire for profits. On the other hand, standards can have the undesirable effect of limiting the safety efforts and investment of companies that feel their legal and moral responsibilities are fulfilled if they follow the standards. Some of the most effective standards and efforts for safety come from users. Manufacturers have more incentive to satisfy customers than to satisfy government agencies.[50]

Professional licensing

Another controversial approach to improving software quality is mandatory licensing of software development professionals. Laws require licenses for hundreds of trades and professions. Licensing requirements typically include specific training, the passing of competency exams, ethical requirements, and continuing education. The desired effect is to protect the public from poor quality and unethical behavior. The history of mandatory licensing in many fields shows that the actual goals and the effects were and are not always very noble. In some trades (plumbing, for example), the licensing requirements were devised to keep black people out. Requirements for specific degrees and training programs, as opposed to learning on one's own or on the job, tend to keep poorer people from qualifying for licenses. Economic analyses have shown that the effect of licensing is to reduce the number of practitioners in the field and keep prices and income for licensees higher than they would otherwise be, in many cases without any improvement in quality.[51] Some see a legal prohibition on working for pay without a government-approved license as a fundamental violation of the freedom to work (i.e., of the negative right, or liberty, to work, in the terms of Section 1.4.3).

Clashes between licensing laws and the Web: Section 3.2.4

These objections do not apply to the many valuable voluntary approaches to measuring or certifying qualifications of software personnel, for example, a diploma from a respected school and certification programs by professional organizations, particularly for advanced training in specialized areas.

Taking responsibility

In some cases of computer errors, businesses pay customers for problems or damages. For example, Intuit offered to pay interest and penalties that resulted from errors in flawed income-tax programs. When United Airlines mistakenly posted ticket prices on its Web site as low as about $25 for flights between the U.S. and Europe, it honored tickets purchased before it corrected the error. United, at first, charged the buyers the correct fare and probably had the legal right to do so, but the airline concluded that having angry customers would cost more than the tickets. We noted that business pressures are often a reason for cutting corners and releasing defective products. Business pressure can also be a cause for insistence on quality and maintaining good customer relations. Good business managers recognize the importance of customer satisfaction and the reputation of the business. Also, some businesses have an ethical policy of behaving responsibly and paying for mistakes, just as a person would pay for accidentally breaking a neighbor's window with a misdirected softball.

Other market mechanisms besides consumer backlash encourage a quality job. Insurance companies have an incentive to evaluate the systems they insure. The insurer for the company that operated several faulty communications satellites commented that it would consider the company's lapse in testing and quality control in the future. Satellite-failure incidents illustrate another market mechanism for dealing with the risk of computer failures: Some businesses paid a higher rate for "uninterrupted" service. That is, the service company would switch their communications quickly to other satellites in case of a failure. Businesses that can withstand a few hours of interruption need not pay for that extra protection. Organizations whose communications are critical to public safety, such as police departments and hospitals, should take responsibility to ensure they have appropriate backup service, possibly paying extra for the higher level of service.

How can customers protect themselves from faulty software? How can a business avoid buying a seriously flawed program? For high-volume, consumer and small-business software, one can consult the many magazines and Web sites that review new programs. Specialized systems with a small market are more difficult to evaluate before purchase. We can check the seller's reputation with the Better Business Bureau. We can consult previous customers and ask how well they did the job. Online user groups for specific software products are excellent sources of information for prospective and current customers. In the case of the Therac-25, the users eventually spread information among themselves. If the Web had existed at the time of the accidents, it is likely that the problems would have been identified sooner and that some of the accidents would not have happened.

8.4 Dependence, Risk, and Progress

8.4.1 ARE WE TOO DEPENDENT ON COMPUTERS?

Comments about our dependence on computers appear in many discussions of the social impact of computers. Because of their usefulness and flexibility, computers are now virtually everywhere. Is this good? Or bad? Or neutral? The word "dependence" often has a negative connotation. "Dependence on computers" suggests a criticism of our society or of our use of computers. Is it appropriate?

In Holland, no one discovered the body of a reclusive, elderly man who died in his apartment until six months after his death. Eventually, someone noticed that he had a large accumulation of mail. This incident was described as a "particularly disturbing example of computer dependency." Many of the man's bills, including rent and utilities, were paid automatically. His pension check went automatically to his bank account. Thus, "all the relevant authorities assumed that he was still alive."[52] But who expects the local gas company or other "relevant authorities" to discover a death? The problem here, clearly, was the lack of concerned family, friends, and neighbors. I happened to be present in a similar situation. An elderly, reclusive woman died in her home. Within two days, not six months, the mailman noticed that she had not taken in her mail. He informed a neighbor, and together they checked the house. I do not know if her utility bills were paid automatically; it is irrelevant.

On the other hand, many people and businesses are not prepared to do without the computer systems they use every day. A BlackBerry e-mail blackout lasted nine hours. It disrupted the work of bankers, technology workers, talent agents, and others who depend on constant communication via their Blackberry—some who receive more than 500 e-mails per day. A physician commented that modern hospitals and clinics cannot function efficiently without medical-information systems. Modern crime fighting depends on computers. Some military jets cannot fly without the assistance of computers. In several incidents, computer failures or other accidents knocked out communication services. Drivers could not buy gasoline with their credit cards. "Customers were really angry," said a gas station manager. Stockbrokers could not connect to New York by telephone or computer. More than 1,000 California state lottery terminals were down; people could not buy tickets or collect winnings. A supermarket manager reported "Customers are yelling and screaming because they can't get their money, and they can't use the ATM to pay for groceries."[53]

Is our "dependence" on computers different from our dependence on electricity, which we use for lighting, entertainment, manufacturing, medical treatments—and just about everything? Is our "dependence" on computers different from a farmer's dependence on a plow? The Sioux people's dependence on their bows and arrows? Modern surgery's dependence on anesthesia?

Computers and plows are tools. We use tools because we are better off with them than without them. They reduce the need for hard physical labor and tedious routine mental labor. They help us be more productive, or safer, or more comfortable. When we have a good tool, we can forget, or no longer even learn, the older method of performing a task. If the tool breaks down, we are stuck. We cannot perform the task until someone fixes it. That can mean that no telephone calls get through for several hours. It might mean the loss of a large amount of money, and it can mean danger or death for some people. But the negative effects of a breakdown do not condemn the tool. To the contrary, for many computer applications (not all), the inconveniences or dangers of a breakdown are a reminder of the convenience, productivity, or safety the tool provides when it is working. It reminds us, for example, of the billions of communications, carrying voice, e-mail, photos, and data, that are possible or more convenient or cheaper because of computers.

We could avoid the risk of a broken plow by plowing with our hands. We could avoid the risk of losing a document file on a disk by doing all our writing by hand on paper. Even ignoring the possibility of a fire destroying our paper records, it should be clear that we reject the "safe" (nondependent) option because, most of the time, it is less convenient and less productive.

Some misconceptions about dependence on computers come from a poor understanding of the role of risk, confusion of "dependence" with "use," and blaming computers for failures where they were only innocent bystanders. On the other hand, abdication of responsibility that comes from overconfidence or ignorance is a serious problem. There are valid technical criticisms of dependence when a system design allows a failure in one component to cause a major breakdown. There are valid criticisms of dependence when businesses, government agencies, and organizations do not make plans for dealing with system failures. The wise individual is grateful for ATMs and credit cards but keeps a little extra cash at home in case they do not work.

8.4.2 RISK AND PROGRESS

Electricity lets us heat our homes, cook our food, and enjoy security and entertainment. It also can kill you if you're not careful.

—"Energy Notes" (Flyer sent with San Diego Gas & Electric utility bills)

We trust our lives to technology every day. We trust older, noncomputer technologies every time we step into an elevator, a car, or a building. As the tools and technologies we use become more complex and more interconnected, the amount of damage that results from an individual disruption or failure increases, and we sometimes pay the costs in dramatic and tragic events. If a person, out for a walk, bumps into another person, neither is likely to be hurt. If both are driving cars at 60 miles per hour, they could be killed. If two jets collide, or one loses an engine, several hundred people could be killed. However, the death rate per mile traveled is lower for air travel than for cars.

Most new technologies were not very safe when first developed. If the death rate from commercial airline accidents in the U.S. were the same now as it was 50 years ago, 8,000 people would die in plane crashes each year. In some early polio vaccines, the virus was not totally inactivated. The vaccines caused polio in some children. We learn how to make improvements. We discover and solve problems. Scientists and engineers study disasters and learn how to prevent them and how to recover from them.

When I first read about the GPWS (the system that warns pilots if they are headed toward a mountain, Section 8.1.5), it occurred to me that if American Airlines had installed it on the plane that crashed near Cali, Colombia, in 1995, they would have saved many lives. Then I read that this crash triggered adoption of the GPWS. No commercial U.S. airliner has crashed into a mountain since then. Similarly, a disastrous fire led to the development of fire hydrants—a way to get water at the scene from the water pipes under the street. Automobile engineers used to design the front of an automobile to be extremely rigid, to protect passengers in a crash. But people died and suffered serious injuries because the car frame transmitted the force of a crash to the people. The engineers learned it was better to build cars with "crumple zones" to absorb the force of impact.[54] Software engineering textbooks use the Cali crash as an example so that future software specialists will not repeat the mistakes in the plane's computer system. We learn.

What has happened to the safety record in other technologies? The number of deaths from motor vehicle accidents in the U.S. declined from 54,633 in 1970 to roughly 42,600 in 2006 (while population and the number of cars, of course, increased).[55] Why? One significant factor is increased education about responsible use (i.e., the campaign against drunk driving). Another is devices that protect people when the system fails (seat belts and airbags). Yet another is systems that help avoid accidents (many of which, like airbags, use microprocessors). Examples of the latter include rear-view cameras that help drivers avoid hitting a child when backing up and "night vision" systems that detect obstacles and project onto the windshield an image or diagram of objects in the car's path. Yet another is electronic stability systems. These systems have sensors that detect a likely rollover, before the driver is aware of the problem, and electronically slow the engine.

As use of technology, automation, and computer systems has increased in virtually all work places, the risk of dying in an on-the-job accident dropped from 39 among 100,000 workers (in 1934) to four in 100,000 in 2004.[56]

Risk is not restricted to technology and machines. It is a part of life. Sharp tools are risky. Someone living in a jungle faces danger from animals. A desert hiker faces rattlesnakes. We are safer if we know the risks and take reasonable precautions. We are never 100% safe.

There are some important differences between computers and other technologies. Computers make decisions; electricity does not. The power and flexibility of computers encourages us to build more complex systems—where failures have more serious consequences. The pace of change in computer technology is much faster than that in other technologies. Software is not built from standard, trusted parts as is the case in many engineering fields. These differences affect the kind and scope of the risks we face. They need our attention as computer professionals, as workers and planners in other fields, and as members of the public.

Observations

We have made several points:

1. Many of the issues related to reliability and safety for computers systems have arisen before with other technologies.
2. Perfection is not an option. The complexity of computer systems makes errors, oversights, and so on likely.
3. There is a *learning curve* for new technologies. By studying failures, we can reduce their occurrence.
4. We should compare risks of using computers with risks of other methods and with benefits obtained.

This does not mean that we should excuse or ignore computer errors and failures because failures occur in other technologies. It does not mean we should tolerate carelessness or negligence because perfection is not possible. It does not mean we should excuse accidents as part of the learning process, and it does not mean we should excuse accidents because, on balance, the contribution of computer technology is positive.

The potential for serious disruption of normal activities and danger to people's lives and health because of flaws in computer systems should always remind the computer professional of the importance of doing his or her job responsibly. Computer system developers and other professionals responsible for planning and choosing systems must assess risks carefully and honestly, include safety protections, and make appropriate plans for shutdown of a system when it fails, for backup systems where appropriate, and for recovery.

Knowing that one will be liable for the damages one causes is strong incentive to find improvements and increase safety. When evaluating a specific instance of a failure, we can look for those responsible and try to ensure that they bear the costs of the damage they caused. It is when evaluating computer use in a particular application area or when evaluating the technology as a whole that we should look at the balance between risks and benefits and compare the risks and benefits with those of alternative technologies.

EXERCISES

Review Exercises

8.1 List two cases described in this chapter in which insufficient testing was a factor in a program error or system failure.

8.2 What are two kinds of computer usage that can cause repetitive strain injury? What are two occupations where repetitive strain injury occurs, but workers are not using computers?

8.3 List two cases described in this chapter in which the provider did an inadequate job of informing customers about flaws in the system.

8.4 What was one cause of the delay in completing the Denver airport?

8.5 What is one case in which reuse of software caused a serious problem?

8.6 Describe one principle of human-interface design that is particularly important in safety-critical applications.

General Exercises

8.7 a) Suppose you write a computer program to add two integers. Assume that each integer and their sum will fit in the standard memory unit the computer uses for integers. How likely do you think it is that the sum will be correct? (If you run the program a million times on different pairs of integers, how many times do you think it would give the correct answer?)

b) Suppose a utility company has a million customers and it runs a program to determine whether any customers have overdue bills. How likely do you think it is that the results of the program will be correct?

c) Probably your answers to parts (a) and (b) were different. Give some reasons why the likely number of errors would be different in these two examples.

8.8 Consider the case described in Section 8.1.2 in which a school assumed a boy was a drug abuser because two schools used different disciplinary codes in their computerized records. Describe some policies or practices that can help prevent such problems.

8.9 A man applied for jobs at several retail stores. They all turned him down. Eventually, he learned that the stores used a database to screen applicants and it listed him as a shoplifter. A real shoplifter had given the police the innocent man's identification from a lost wallet.

Would this incident have been as likely to happen 30 years ago? Why or why not? The innocent man sued the company that maintains the database and the store where the real shoplifter was arrested. Should he win? Give reasons.

8.10 In response to the lack of scientific knowledge about whether or how computer keyboards cause RSI, a plaintiff's lawyer who handled more than 1000 RSI lawsuits commented that "The law can't wait on science."[57] Give some arguments to support this statement (as it applies to RSI lawsuits). Give some arguments against it. Do you agree with the statement? Why?

8.11 List several possible reasons why a car rental company might mistakenly list one of its rented cars as stolen. (See page 409.) Which of these could better software or better policies prevent? Which are the kinds of mistakes that would be difficult or impossible to prevent?

8.12 Suppose you are responsible for hiring drivers for a trucking company. The FBI has provided a terrorist watch list to the company. Develop a policy for how to use the list.

8.13 Consider the standardized-test score reporting error discussed in the box in Section 8.1.3. Suppose the test company had reported scores to the schools as significantly higher, rather than lower, than the correct scores. Do you think the schools would have questioned the scores? Do you think anyone would have discovered the error? If so, how? Give a few examples of situations where you think people would not report computer errors. For each example, give your reason (e.g., optimism, ignorance, gullibility, dishonesty, others).

8.14 The U.S. Immigration and Naturalization Service (INS) sent visa approval notifications for two of the September 11 hijackers six months after they crashed airplanes into the World Trade Center. No one had updated the INS database to cancel the visas (which the INS had approved before September 11). This incident generated a lot of publicity embarrassing for the INS. What policy or process could have avoided this error? Is it reasonable to expect that the INS should have prevented it?

8.15 In Section 8.3.1 we gave examples of problems that occur when data in a database are not up to date. The examples involved law enforcement databases. Give an example involving a business database. (If you do not know of an actual incident, describe a reasonable, hypothetical one.)

8.16 Many college students attend several colleges before they eventually graduate. It would be a convenience for students if they could order a complete transcript (say, for job applications) from the federal student database discussed in Section 2.2.1*. Describe several ways in which getting transcripts from the database might be riskier than getting them from the individual colleges.

8.17 Suppose you are on a consulting team to design a computerized voting system for your state. People will vote on computers at the voting place (not over the Internet; we considered Web-based voting in an exercise in Chapter 5). What are some important design considerations?

8.18 You are the traffic manager for a small city. The City Council has directed you to buy and install a computer system to control the traffic lights. Its main purpose is to adjust the timing of the lights to improve traffic flow at rush hours and for special events.
a) List some potential risks of the system.
b) List some technical requirements and/or specifications you would put in the proposal for safety.

8.19 Find several provisions of the Software Engineering Code of Ethics and Professional Practice (Appendix A.1) that were violated in the Therac-25 case.

8.20 Several models of a medical infusion pump in use worldwide have a defect called *key-bounce*. When a user types the dosage on the keypad, a key pressed once could bounce and cause the digit to record twice. Thus a dose of two units might become 22 units. The pump could give a patient an overdose of drugs. The company was warned of problems with the pumps in the late 1990s. The Food and Drug Administration seized a supply of the pumps from the manufacturer in 2006 and issued a recall notice.[58]

Identify several things that various people did, or probably did, that were wrong.

8.21 Suppose you are responsible for the design and development of a computer system to control an amusement-park ride. Sensors in the seats will determine which seats are occupied, so the software can consider weight and balance. The system will control the speed and time of the ride. The amusement park wants a system where, once the ride starts, a person is not needed to operate it.

List some important things that you can or should do to ensure the safety of the system. Consider all aspects of development, technical issues, operating instructions, and so on.

8.22 After making a programming change in a major bank's computer system, an employee forgot to enter certain commands. As a result, approximately 800,000 direct deposits received by the bank were not posted to the customer accounts until the next day. What are some potential consequences of the error? If you were the bank president, what would you say in a statement to the news media or your customers?

8.23 Who are the "good guys"? Pick two people or organizations mentioned in this chapter whose work helped make systems safer or reduced the negative consequences of errors. Tell why you picked them.

8.24 We mentioned that some cell phones contain a few million lines of computer code. Estimate how many pages one million lines of code would take up if printed. (State your assumptions.)

*The actual database proposal does not include providing services such as transcripts for individual students.

8.25 At some hospitals, doctors use a computer system to order drugs for patients. The system eliminates errors from reading doctors' handwriting, and it automatically checks for conflicts with other medicines the patient is taking. Doctors sometimes do not log out after using a terminal. When another doctor uses the same terminal, the system assigns drugs ordered by the second doctor to the first doctor's patient. Describe two features that such systems could include to reduce this kind of error.

8.26 A technician on a Navy guided-missile ship entered a zero in the wrong place in a computer program calibrating a fuel valve. The program divides another number by the entered number. It crashed because division by zero is an invalid operation. The program failure caused the ship's entire Local Area Network to fail, leaving the ship dead in the water for almost three hours.

To what degree is each of the following people responsible: the technician, the person who wrote the fuel-valve calibration program, the person who selected and purchased the ship's Local Area Network, the software company that sells the network software, the captain of the ship? What, if anything, did each do wrong, and what could reduce the chance of such a problem in the future? Are there any other people who bear some of the responsibility? (You cannot give a full and definite answer without more detailed information. Where necessary, indicate what additional information you need and how it would affect your answer.)

8.27 A computer error in a contest sponsored by a multinational beverage company caused distribution of 800,000 winning numbers instead of the intended 18. The face value of the winnings amounted to $32 billion. Suppose you are an employee on a team given the task of deciding how to respond to this problem. Make some suggestions.

8.28 The Food and Drug Administration maintains a registry of more than 120,000 drugs. An investigation by the Department of Health and Human Services found that the information on about 34,000 drugs was incorrect or out of date. Nine thousand drugs were missing from the directory.[59] Describe several possible risks of the database being so out of date. Give as many possible reasons as you can think of why the database was out of date.

8.29 There is a story that a major retail company "lost" a warehouse from its inventory computer system for three years. The warehouse received no goods and shipped none. A separate system handled payroll, so the employees continued to get paychecks. To what extent is this a computer failure? What other important factors are part of the problem?

8.30 Choose a noncomputer activity that you are familiar with and that has some risks (e.g., skateboarding, scuba diving, or working in a restaurant). Describe some of the risks and some safety practices. Describe analogies with risks related to computer systems.

8.31 Software developers are sometimes advised to "design for failure." Give some examples of what this might mean.

8.32 Assume you are a professional working in your chosen field. Describe specific things you can do to reduce the impact of any two problems we discussed in this chapter. (If you cannot think of anything related to your professional field, choose another field that might interest you.)

8.33 Think ahead to the next few years and describe a new problem, related to issues in this chapter, likely to develop from computing technology or the Web.

Assignments

These exercises require some research or activity.

8.34 Read a few items in the current issue of the *Risks Digest* (www.csl.sri.com/users/risko/risks.txt). Write a summary of two items.

8.35 Find out which (if either) of the following views is common among eye doctors: (1) Working at a computer screen for many hours permanently weakens vision. (2) Eye strain from computer use is temporary; rest breaks, ergonomic changes, or similar techniques can relieve it.

Class Discussion Exercises

These exercises are for class discussion, perhaps with short presentations prepared in advance by small groups of students.

8.36 Assume that the family of one of the victims of the Therac-25 has filed three lawsuits. They are suing a hospital that used the machine, the company that made the machine (AECL), and the programmer who wrote the Therac-25 software. Divide students into six groups: attorneys for the family against each of the three respondents and attorneys for each of the three respondents. Each group is to present a five-minute summation of arguments for its case. Then, let the class discuss all aspects of the case and vote on the degree of responsibility of each of the respondents.

8.37 Consider the following scenario. A state's highway patrol keeps records of stolen cars in its computer system. There is no routine process for updating records when stolen cars are recovered. The system still listed a car as stolen a few years after it had been recovered and the owner sold it. A highway patrol officer shot and killed the new owner of the car during a traffic stop. The officer thought the car was stolen and that the driver was acting suspiciously. An investigation concluded that the officer "acted in good faith."* To what extent should the error in the database affect the family's wrongful-death lawsuit against the highway patrol. Suggest a feature the database should have that might prevent such incidents.

8.38 A factory contains an area where robots do all the work. There is a fence around the area. Human workers are not supposed to enter while the robots are working. When anyone opens the gate in the fence, it automatically cuts off power to the robots. A worker jumped over the fence to repair a robot that was malfunctioning. Another robot, bringing parts to the malfunctioning robot, accidentally pinned the worker against a machine, killing him.

Suppose your class is a consulting team hired (by a neutral party) to investigate this case and write a report. Consider several factors relevant in safety-critical systems. What was done right? What was done wrong? Is there important information not included in this summary of the case that you would ask about? If so, what? What degree of blame do you assign to the software company that designed the robot system, the company that operates the factory, and the worker? Why? What changes, if any, should the factory operators make to reduce the likelihood of more deaths?

NOTES

1. Robert N. Charette, "Why Software Fails," *IEEE Spectrum*, September 2005, www.spectrum.ieee.org/sep05/1685 (accessed December 8, 2006).
2. *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*, catless.ncl.ac.uk/risks (accessed September 7, 2007).
3. Philip E. Ross, "The Day the Software Crashed," *Forbes*, 153, no. 9 (April 25, 1994), pp. 142–156; Peter G. Neumann, "Inside Risks," *Communications of the ACM*, July 1992, p. 122.
4. Andrea Robinson, "Firm: State Told Felon Voter List May Cause Errors," *Miami Herald*, February 17, 2001.

*I have changed some details, but this scenario was suggested by a real case.

5. Dan Joyce, e-mail correspondence, May 17, 1996 (the adoption case). Study by the Office of Technology Assessment, reported in Rothfeder, *Privacy for Sale*; "Jailing the Wrong Man," *Time*, February 25, 1985, p. 25; David Burnham, "Tales of a Computer State," *The Nation*, April 1983, p. 527; Evelyn Richards, "Proposed FBI Crime Computer System Raises Questions on Accuracy, Privacy," *Washington Post*, February 13, 1989, p. A6; "Wrong Suspect Settles His Case of $55,000," *New York Times*, March 6, 1998, p. 30; Peter G. Neumann, "Risks to the Public in Computer and Related Systems," *Software Engineering Notes*, 13, no. 2 (April 1988), p. 11. Several similar cases are reported by Peter G. Neumann in "Inside Risks," *Communications of the ACM*, January 1992, p. 186; Herb Caen, *San Francisco Chronicle*, July 25, 1991.
6. Ann Davis, "Post-Sept. 11 Watch List Acquires Life of Its Own," *Wall Street Journal*, November 19, 2002, p. A1.
7. "Terrorist Watch List Screening: Efforts to Help Reduce Adverse Effects on the Public," GAO-06-1031, September 29, 2006, p. 34, www.gao.gov/new.items/d061031.pdf (accessed December 11, 2006).
8. *Arizona v. Evans, reported in* "Supreme Court Rules on Use of Inaccurate Computer Records," *EPIC Alert*, March 9, 1995.
9. Quoted in Associated Press, " 'BlackBerry Thumb' on the Rise," CBS News, October 21, 2005, www.cbsnews.com/stories/2005/10/21/tech/main960825.shtml (accessed December 11, 2006).
10. Edward Felsenthal, "An Epidemic or a Fad? The Debate Heats up over Repetitive Stress," *Wall Street Journal*, July 14, 1994, p. A1; R. L. Linscheid and J. H. Dobyns, "Athletic Injuries of the Wrist," *Clinical Orthopedics*, September 1985, pp. 141–151; *American Annals of the Deaf* ; David M. Rempel, Robert J. Harrison, and Scott Barnhart, "Work-Related Cumulative Trauma Disorders of the Upper Extremity," *Journal of the American Medical Association*, 267, no. 6 (February 12, 1992), pp. 838–842.
11. Frederic M. Biddle, John Lippman, and Stephanie N. Mehta, "One Satellite Fails, and the World Goes Awry," *Wall Street Journal*, May 21, 1998, p. B1.
12. Reuters, "Glitch Closes Tokyo Stock Exchange," *The New Zealand Herald*, November 2, 2005, nzherald.co.nz (accessed December 12, 2006); David Craig, "NASDAQ Blackout Rattles Investors," *USA Today, July 18*, 1994, p. 2B; Associated Press, "NASDAQ Defends Its System after Stock-Pricing Errors," *New York Times*, September 13, 1994, p. D19; "Note to Readers," *Boston Globe*, October 25, 1994, p. 52; Julia Flynn, Sara Calian, and Michael R. Sesit, "Computer Snag Halts London Market 8 Hours," *Wall Street Journal*, April 6, 2000, p. A14.
13. mars.jpl.nasa.gov/msp98/orbiter (accessed December 19, 2006).
14. Thomas Hoffman, "NCR Users Cry Foul Over I Series Glitch," *Computerworld*, February 15, 1993, p. 72; Milo Geyelin, "Faulty Software Means Business for Litigators," *Wall Street Journal*, January 21, 1994, p. B1; Milo Geyelin, "How an NCR System for Inventory Control Turned into a Virtual Saboteur," *Wall Street Journal*, August 8, 1994, pp. A1, A5; Mary Brandel and Thomas Hoffman, "User Lawsuits Drag on for NCR," *Computerworld*, August 15, 1994, p. 1.
15. Jacques Steinberg and Diana B. Henriques, "When a Test Fails the Schools, Careers and Reputations Suffer," *New York Times*, May 21, 2001, pp. A1, A10–A11.
16. Ed Felton, "Hotel Minibar Keys Open Diebold Voting Machines," September 18, 2006, www.freedom-to-tinker.com/?p=1064 (accessed December 6, 2006).
17. The DIA delay was widely reported in the news media. A few of the sources I used for the discussion here are Kirk Johnson, "Denver Airport Saw the Future. It Didn't Work," *New York Times*, August 27, 2005, www.nytimes.com (accessed December 12, 2006); W. Wayt Gibbs, "Software's Chronic Crisis," *Scientific American*, 271, no. 3 (September 1994), pp. 86–95; Robert L. Scheier, "Software Snafu Grounds Denver's High-Tech Airport," *PC Week*, 11, no. 19 (May 16, 1994), p. 1; Price Colman, "Software Glitch Could Be the Hitch. Misplaced Comma Might Dull Baggage System's Cutting Edge," *Rocky Mountain News*, April 30, 1994, p. 9A; Steve Higgins, "Denver Airport: Another Tale of Government High-Tech Run Amok," *Investor's Business Daily*, May 23, 1994, p. A4; Julie Schmit, "Tiny Company Is Blamed for Denver Delays," *USA Today*, May 5, 1994, pp. 1B, 2B.
18. Scheier, "Software Snafu Grounds Denver's High-Tech Airport."
19. Wayne Arnold, "How Asia's High-Tech Airports Stumbled," *Wall Street Journal*, July 13, 1998, p. B2.
20. Charette, "Why Software Fails"; Virginia Ellis, "Snarled Child Support Computer Project Dies," *Los Angeles Times*, November 21, 1997, pp. A1, A28; Peter G. Neumann, "System Development Woes," *Communi-cations of the ACM*, December 1997, p. 160; Harry Goldstein, "Who Killed the Virtual Case File?" *IEEE Spectrum*, September 2005, www.spectrum.ieee.org/sep05/1455 (accessed December 12, 2006).
21. Charette, "Why Software Fails."
22. H. Travis Christ, quoted in Linda Rosencrance, "US Airways Partly Blames Legacy Systems for March Glitch," *Computerworld*, March 29, 2007; Linda Rosencrance, "Glitch at U.S. Airways Causes Delays," *Computerworld*, March 5, 2007, www.computerworld.com (accessed April 24, 2007).

23. "Airbus Safety Claim 'Cannot Be Proved,' " *New Scientist*, 131, no. 1785 (September 7, 1991), p. 30; Robert Morrell Jr., *Risks Forum Digest*, 16, no. 20 (July 6, 1994); "Training 'Inadequate' Says A320 Crash Report," *Flight International*, December 22, 1993, p. 11 (on the January 1992 Strasbourg A320 crash, excerpted by Peter B. Ladkin in *Risks Forum Digest*, January 2, 1994); Ross, "The Day the Software Crashed," p. 156, on the 1993 Warsaw crash; David Learmont, "Lessons from the Cockpit," *Flight International*, January 11, 1994.
24. Alan Levin, "FAA Finally Unveils New Radar System," *USA Today*, January 20, 1999, p. 01. A; Anna Wilde Mathews and Susan Carey, "Airports Have Delays, Cancellations Due to Problems in Air-Traffic Control," *Wall Street Journal*, May 7, 1999, p. A20; "Software Glitch" (editorial), *San Diego Union Tribune*, October 23, 2000, p. B8; Robert Fox, "News Track: Air Communication Breakdown," *Communications of the ACM*, 43, no. 8 (August 2000), p. 10.
25. Raju Narisetti, Thomas E. Weber, and Rebecca Quick, "How Computers Calmly Handled Stock Frenzy," *Wall Street Journal*, October 30, 1997, pp. B1, B7; Peter G. Neumann, "System Development Woes," *Communications of the ACM*, December 1997, p. 160.
26. Alan Levin, "Airways Are the Safest Ever," *USA Today*, June 29, 2006, pp. 1A, 6A; William M. Carley, "New Cockpit Systems Broaden the Margin of Safety for Pilots," *Wall Street Journal*, March 1, 2000, pp. A1, A10.
27. FAA Commissioner Marion Blakey, "Keeping Pace with Change," speech at the National Business Aviation Association, Orlando, FL, October 17, 2006, www.faa.gov/news/speeches/news_story.cfm?newsId=7439 (accessed December 19, 2006).
28. Carley, "New Cockpit Systems Broaden the Margin of Safety for Pilots"; Barry H. Kantowitz, "Pilot Workload and Flightdeck Automation," in M. Mouloua and R. Parasuraman, eds., *Human Performance in Automated Systems: Current Research and Trends* (Lawrence Erlbaum, 1994), pp. 212–223.
29. Andy Pasztor, "Airbus to Use Computers for Avoiding Collisions," *Wall Street Journal Europe*, May 29, 2006, p. 5.
30. "FDA Statement on Radiation Overexposures in Panama," www.fda.gov/cdrh/ocd/panamaradexp.html (accessed January 4, 2007); Deborah Gage and John McCormick, "We Did Nothing Wrong," Baseline, March 4, 2004, www.baselinemag.com/article2/0,1397,1543564,00.asp (accessed January 2, 2007).
31. Nancy G. Leveson and Clark S. Turner, "An Investigation of the Therac-25 Accidents," *IEEE Computer*, 26, no. 7 (July 1993), pp. 18–41; Jonathan Jacky, "Safety-Critical Computing: Hazards, Practices, Standards, and Regulation," in Charles Dunlop and Rob Kling, eds., *Computerization and Controversy* (Academic Press, 1991), pp. 612–631. Most of the factual information about the Therac-25 incidents in this chapter is from Leveson and Turner.
32. Conversation with Nancy Leveson, January 19, 1995.
33. Jacky, "Safety-Critical Computing," p. 615; Peter G. Neumann, "Risks to the Public in Computers and Related Systems," *Software Engineering Notes*, 16, no. 2 (April 1991), p. 4.
34. Ted Wendling, "Lethal Doses: Radiation that Kills," *Cleveland Plain Dealer*, December 16, 1992, p. 12A. (I thank my student Irene Radomyshelsky for bringing this article to my attention.)
35. *Guns, Germs, and Steel: The Fates of Human Societies* (W. W. Norton, 1997), p. 157.
36. Charette, "Why Software Fails."
37. "Airbus Safety Claim 'Cannot Be Proved,' " p. 30.
38. The report of the inquiry into the explosion is at sunnyday.mit.edu/accidents/Ariane5accidentreport.html (accessed September 12, 2007).
39. *Privacy Journal*, August 1998, p. 4. The database is the Interstate Identification Index.
40. Marisa Taylor, "U.S. Struggles to Keep Tabs on Foreign Visitors Leaving," *San Diego Union-Tribune*, September 16, 2006, p. A7.
41. From an e-mail advertisement for Nancy G. Leveson, *Safeware: System Safety and Computers* (Addison Wesley, 1995).
42. A particularly good article discussing human factors and the causes of the crash is Stephen Manes, "A Fatal Outcome from Misplaced Trust in 'Data,' " *New York Times*, September 17, 1996, p. B11.
43. Kantowitz, "Pilot Workload and Flightdeck Automation," pp. 212–223.
44. Feynman, *What Do You Care What Other People Think?* The shuttle was not immune from problems. The *Risks Digest* includes reports of computer failures caused by a loose piece of solder, subtle timing errors, and other factors.
45. "AT&T Crash, 15 Jan 90: The Official Report."
46. Feynman, *What Do You Care What Other People Think?* pp. 190–194, 232–236. Aeronautics and Space Engineering Board, National Research Council, *An Assessment of Space Shuttle Flight Software Development Processes* (National Academy Press, 1993).
47. Mary Brandel and Thomas Hoffman, "User Lawsuits Drag on for NCR," *Computerworld*, August 15, 1994, p. 1.
48. Professor Philip Koopman of Carnegie Mellon University discusses these issues and others about liability for embedded software in articles posted on his Web site, www.ece.cmu.edu/~koopman/ucita.

49. These problems and trade-offs occur often with regulation of new drugs and medical devices, regulation of pollution, and various kinds of safety regulation. They are discussed primarily in journals on the economics of regulation.
50. Leveson and Turner, "An Investigation of the Therac-25 Accidents," p. 40.
51. See, for example, Walter Williams, *The State Against Blacks* (McGraw-Hill, 1982), Chapters 5–7. One year during a construction lull, a state failed everyone who took the building contractor's license exam. It is illegal in 48 states for most software engineers to call themselves software engineers because of licensing laws for engineers. One company had to spend thousands of dollars changing job titles, business cards, and marketing literature to remove the word "engineer." [Julia King, "Engineers to IS: Drop that Title!" *Computerworld*, 28, no. 22 (May 30, 1994), pp. 1, 119.]
52. Tom Forester and Perry Morrison, *Computer Ethics: Cautionary Tales and Ethical Dilemmas in Computing*, 2nd ed. (MIT Press, 1994), p. 4.
53. Heather Bryant, an Albertson's manager, quoted in Penni Crabtree, "Glitch Fouls up Nation's Business," *San Diego Union-Tribune*, April 14, 1998, p. C1; Miles Corwin and John L. Mitchell, "Fire Disrupts L. A. Phones, Services," *Los Angeles Times*, March 16, 1994, p. A1.
54. An excellent "Nova" series, "Escape! Because Accidents Happen," aired February 16 and 17, 1999, shows examples from 2000 years of history of inventing ways to reduce the injuries and deaths from fires and from boat, car, and airplane accidents.
55. "2006 Traffic Safety Annual Assessment—A Preview" (DOT HS 810 791), National Highway Traffic Safety Administration, July 2007.
56. U.S. Census Bureau, *The 2007 Statistical Abstract*, Table 638; "Workers Killed or Disabled on the Job: 1970 to 2004" (accessed December 19, 2006). "A Fistful of Risks," *Discover*, May 1996, p. 82.
57. Steven Philips, quoted in Felsenthal, "An Epidemic or a Fad?"
58. "Class 1 Recall: Cardinal Health Alaris ® SE Infusion Pumps," Food and Drug Administration, August 10, 2006, www.fda.gov/cdrh/recalls/recall-081006.html (accessed September 7, 2007); Jennifer Corbett Dooren, "Cardinal Health's Infusion Pump Is Seized Because of Design Defect," *Wall Street Journal*, August 29, 2006, p. D3.
59. "FDA Proposes Rules for Drug Registry," *Wall Street Journal*, August 24, 2006, p. D6.

BOOKS AND ARTICLES

- Collins, W. Robert, Keith W. Miller, Bethany J. Spielman, and Phillip Wherry, "How Good Is Good Enough?" *Communications of the ACM*, 37, no. 1 (January 1994): 81–91. A discussion of ethical issues about quality for software developers.
- Dempsey, Paul Stephen, Andrew R. Goetz, and Joseph S. Szyliowicz. *Denver International Airport: Lessons Learned*, McGraw-Hill, 1997.
- Epstein, Richard. *The Case of the Killer Robot*. John Wiley and Sons, 1996.
- Feynman, Richard P. *What Do You Care What Other People Think?* W. W. Norton & Co., 1988. Includes Feynman's report on the investigation of the explosion of the Challenger space shuttle, with many insights about how to, and how not to, investigate a system failure.
- Jacky, Jonathan. "Safety-Critical Computing: Hazards, Practices, Standards, and Regulation," in Charles Dunlop and Rob Kling, eds., pp. 612–631, *Computerization and Controversy*, Academic Press, 1991.
- Leveson, Nancy G. *Safeware: System Safety and the Computer Age*. Addison Wesley, 1995.
- Leveson, Nancy G., and Clark S. Turner. "An Investigation of the Therac-25 Accidents." *IEEE Computer*, 26, no. 7 (July 1993): 18–41.
- Neumann, Peter G. *Computer-Related Risks*. Addison-Wesley, 1995.

- Nielsen, Jakob. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, 2000.
- Norman, Donald. *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. MIT Press, 1998.
- Norman, Donald. *The Psychology of Everyday Things*. Basic Books, 1988. A study of good and bad user interfaces on many everyday devices and appliances.
- Pascarelli, Emil, and Deborah Quilter. *Repetitive Strain Injury: A Computer User's Guide*. John Wiley & Sons, Inc., 1994.
- Peterson, Ivars. *Fatal Defect: Chasing Killer Computer Bugs*. Times Books (Random House), 1995.
- Petrovski, Henry. *To Engineer Is Human: The Role of Failure in Successful Design*. St. Martin's Press, 1985. This book is more about engineering in general, not computer systems design, but the principles and lessons carry over.
- Pfleeger, Shari L., and Joanne Atlee, *Software Engineering: Theory and Practice*. 3rd ed. Pearson Prentice Hall, 2005.
- Rothfeder, Jeffrey. *Privacy for Sale*. Simon & Schuster, 1992. Although the main focus of this book is privacy, it contains many examples of problems that resulted from errors in databases.
- Shneiderman, Ben, and Catherine Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 4th ed. Addison Wesley Longman, 2004.
- Tufte, Edward. *Envisioning Information*. Graphics Press, 1990.
- Tufte, Edward. *Visual Explanations*. Graphics Press, 1997.
- Wildavsky, Aaron. *Searching for Safety*. Transaction Books, 1988. On the role of risk in making us safer.

ORGANIZATIONS AND WEB SITES

- Peter G. Neumann, moderator, *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*: catless.ncl.ac.uk/risks