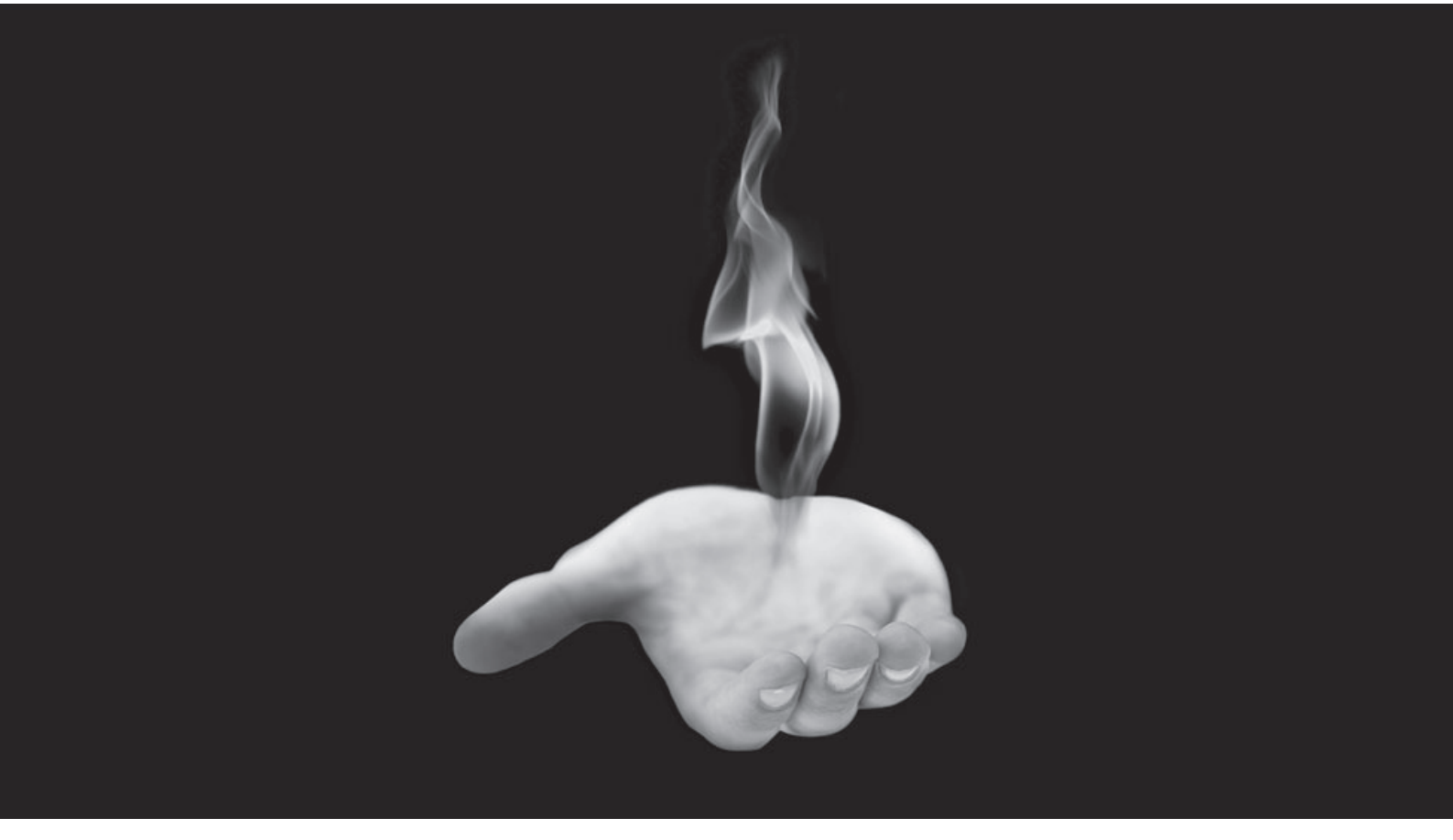
8

Errors, Failures, and Risks



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. Studying these failures and risks contributes to understanding their causes and helps prevent future failures.

Are computer systems too unreliable and too unsafe to use? Or, like many news stories, do the headlines and horror stories emphasize the bad news—the dramatic but unusual events? We hear reports of car crashes, 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: 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.

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 system or application 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 the problems from the perspective of several of the roles we play:

- *A computer user.* Whether we use our own tablet computer or a sophisticated, specialized system at work, we should understand the limitations of computer systems and the need for proper training and responsible use.
- *A computer professional.* If you are planning a career as computer professional (system designer, programmer, or quality assurance manager, for example), studying computer system failures should help you become a better professional. Understanding the source and consequences of 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 have surgery performed by a robot. 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 others. 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 safety-critical 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 professional practices and other approaches to preventing failures and handling them properly when they occur. Section 8.4 puts the risks in perspective in various ways.

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, software engineering journals, and in The Risks-Forum 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

Billing errors

The first few errors we look at are relatively simple ones whose negative consequences were undone with relative ease.

- 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 5000 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 3000 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 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 generate a list of cases for a person 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.

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 pretty good. It is better

than a 99.999% accuracy rate. Is that reasonable? At some point, the expense of improving a system is not worth the gain, especially for applications where the impact of the error is small and errors can be detected (once they occur) and corrected at much lower cost than it would take to try to prevent them.

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. 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. 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. 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.

It is difficult to get accurate and meaningful error rates for major databases with information about millions of people. Also, 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 they indicate that a high percentage of credit records have serious errors.

Errors affecting job applicants: Section 6.3.1

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 their 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 man murdered two men in Washington state after getting their addresses from the state's sex offender database, and another man killed two men listed in Maine's online registry. One of them was in the database because, as a teenager, he had sex with his girlfriend who was a few weeks below the age of consent. While technically not an error in the database, 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.

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 really had rented the rental car they were driving. The car rental company had listed the car as stolen. An adoption agency ran a routine check on an applicant and found a conviction for grand larceny. In fact, the applicant had taken 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. It is understandable that the FBI's National Crime Information Center (NCIC) database showed the innocent man as wanted—someone using his name was committing crimes. However, the innocent man was arrested five times; the database was not corrected. 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 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.)[3]

After the terrorist attacks in 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 emailed 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.[4] Even if someone corrects an error in the original database, problems may not be over for the affected person. Copies of incorrect or mislabeled data remain in other systems.

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

* AWOL means "absent without official leave."

- 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 due to carelessness) in data entry
- Failure to update information and correct errors
- Lack of accountability for errors

The first factor 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. The remaining factors in the list above are all within our control as individuals, professionals, and policy makers. We discuss them throughout this chapter.

> *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[5]

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 give examples of failures, with some 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 and testing, of having backup plans in case of failures, and of honesty in dealing with errors apply to large projects in other professions as well.

Millions of BlackBerry users did not get their email for nine hours after the company installed a faulty software update. Customers of AT&T lost telephone service for voice and data for hours because of a software error in a four-million-line program. A three-line change in a two-million-line telecommunications switching program caused a failure of telephone networks in several major cities. Although the program underwent 13 weeks of testing, it was not retested after the change—which contained a typo. 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. Log-ins overloaded Skype's peer-to-peer network system when a huge number of people rebooted

BlackBerry thumb and RSI

Millions of children play games on small electronic devices, and millions of adults answer email on portable electronic gadgets with mini keypads. In many professions, people type on a 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 by a variety of 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, or secretaries 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.)[6] Computer game players and smartphone and keyboard users 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. 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. 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. But 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 for electronic devices, as they might, 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 tweeters 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)

their computers after installing routine Windows updates. A majority of Skype's Internet phone users could not log in for two days.

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 United States, including hospitals and police departments. 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.[7]

Every few years, the computer system of one of the world's large stock exchanges or brokerages fails. An error in a software upgrade shut down trading on the Tokyo Stock Exchange. 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.[8]

A failure of Amtrak's reservation and ticketing system during Thanksgiving weekend caused delays because agents had no printed schedules or fare lists. Virgin America airline switched to a new reservation system a month before Thanksgiving. Its website and check-in kiosks did not work properly for weeks.[9]

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.[10]

An inventory management system caused severe losses for businesses that used it. The system had been developed and written for one computer and operating system, then modified and sold to run on another. The modified system sometimes did not accept purchase orders, causing an expensive backlog in orders. Printing invoices took minutes instead of seconds. The system gave incorrect information about inventory and prices. Several users claimed that although the company that sold the system received complaints of serious problems from many customers, the company told customers the problems they were having were unique. Eventually, the company agreed it "did not service customers well" and the program should have undergone more extensive testing. The sources of the problems included technical difficulties (converting software to a different system), poor management decisions (inadequate testing of the modified system on the new platform), and, according to the customers, dishonesty in promoting the system and responding to the problems.[11]

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

Destroying careers and summer vacations[12]

CTB/McGraw-Hill develops and scores standardized tests for schools. Millions of 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 9000 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 an 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. In a case with a similar lesson, Florida state officials relied on computer-generated lists of possible felons to prevent some people from voting, even though the database company supplying the lists said the state should do additional verification.[13] Relying solely on one factor or on data from one database is temptingly easy. It is a temptation that people responsible for critical decisions in many situations should resist.

ballots sometimes difficult to read or ambiguous. Recounting was a slow tedious process. Many people saw electronic systems as the solution. In 2002, Congress passed the Help America Vote Act and authorized $3.8 billion to improve voting systems. By the 2006 elections, only a very small percentage of Americans voted with paper ballots. The rush to electronic voting machines demonstrated that they too could have numerous faults. Here are some of the problems that occurred: 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 4000 votes because the machine's memory was full. A programming error generated 100,000 extra votes in one Texas county. A programming error caused some candidates to receive votes actually cast for other candidates.

Security against vote fraud and sabotage is a significant issue in elections. 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 electronic voting machines. They said the machines had insecure encryption techniques (or none at all), insufficient security for installation of upgrades to software, and poor physical protection of the memory card on which the system stores votes. One research group demonstrated a system's vulnerability to a virus that essentially took over the machine and manipulated the vote results. They found that voting system developers lacked sufficient security training. Programmers omitted basic procedures such as input validation and boundary checks. 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. There were certification standards for voting systems, but some flawed systems were certified; the standards were inadequate.[14] 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 task of training users is complex. Thousands of ordinary people volunteer as poll workers and manage and operate the machines on election day.) An underlying cause is haste. In projects like these, the desire of states to obtain federal grants encourages haste. The grants have short limits on how soon the states must spend the money,

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 introduce a host of other problems that must be addressed. The first step is to recognize that developing them requires a high degree of professionalism and a high degree of security. In the near future, we will probably vote online. Sadly, it is likely that, at least at first, online voting systems will be highly vulnerable to fraud.

> *Those who cast the votes decide nothing. Those who count the votes decide everything.*
>
> —Attributed to Joseph Stalin (Premier of the Soviet Union)[15]

Stalled airports: Denver, Hong Kong, and Malaysia

Ten months after the $3.2 billion Denver International Airport airport was supposed to have opened, I flew over the huge airport. It 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 miles of wide highway leading to it. 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.[16]

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 10 minutes via 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 4000 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 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:

- *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

* 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.

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."[17] 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.

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 tropical heat.

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 the system paralyzed it. "There's nothing wrong with the system," said a spokeman at the airport in Malaysia. 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.[18]

Abandoned systems

The flaws in some systems are so extreme 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

- 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 Why abandonned systems failed.

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. After spending $4 billion, the IRS abandoned a tax-system modernization plan; a Government Accountability Office 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. (The FBI's next major attempt at a paperless case-management system was scheduled for completion in 2009 but delayed at least until 2012.)[19] There are many more such examples.

Software expert Robert Charette estimates that from 5% to 15% of information technology projects are abandoned before or soon after delivery as "hopelessly inadequate." Figure 8.1 includes some reasons he cites.[20] 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 reservations 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 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.[21] 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. Old programs often had little or no documentation, and the programmers who wrote the software or operated the systems have left the company, retired, or died. 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."

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. It would require a major retraining project. 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 What Goes Wrong?

Computer systems fail for two general reasons: the job they are doing is inherently difficult, and sometimes the job is done poorly. Several factors combine to make the task difficult. Computer systems 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. Automobiles, passenger airplanes, and jet fighters contain millions of lines of computer code.[22] A smartphone has several millions of lines of code. Computer software is "nonlinear" in the sense that, whereas a small error in a mechanical system 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 house, a car, or any complex system.) 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.

- 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
  Confusing user interfaces
  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; 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.

Overconfidence

Overconfidence, or an unrealistic or inadequate understanding of the risks in a complex system, is a core issue. When system developers and users appreciate the risks, 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 people do not back up their files or contact lists until after their computers crash or they lose their phones?

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, fallen leaves on train tracks, a cup of coffee spilled in an airplane cockpit, and so on).

Unrealistic estimates of reliability or safety can come from genuine lack of understanding, from carelessness, or from intentional misrepresentation. People without a high regard for honesty, or who work in an organization that lacks a culture of honesty and focus on safety, 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 first launch of France's 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.[23] 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) that the Transportaton Security Agency had given to the airlines. 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.

Do these examples tell us that we should not reuse software? One of the goals of programming paradigms such as 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.

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,

and devices that increase the safety of surgeries. 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 more than 100 times the intended dose. 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.[24] 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.[25])

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 the intended treatment, 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.

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 developers apparently assumed the software functioned 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 an 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, non-safety-related errors. Yet one-key resumption occurred in some of the accidents in which patients received multiple overdoses.

Atomic Energy of Canada, Ltd. (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. Because some readers have little or no programming knowledge, 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 set-up 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 might 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. Medical facilities 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 after that with no new incidents of overdoses.[26]

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. 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 emphasize that safety requires more than bug-free code, we consider failures and accidents involving other radiation treatment systems. 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.[27] Two news reporters reviewed more than 4000 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.* 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.)[28]

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

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

examples remind us 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[29]

8.3.1 Professional Techniques

The New York Stock Exchange installed a $2 billion system with hundreds of computers, 200 miles of fiber-optic cable, 8000 telephone circuits, and 300 data routers. The exchange managers prepared for spikes in trading by testing the system on triple and quadruple the normal trading volume. On one day, the exchange processed 76% more trades than the previous record. The system handled the sales without errors or delays.[30] We have been describing failures throughout this chapter. Many large, complex computer systems work extremely well. We rely on them daily. How can we design, build, and operate systems that are likely to function well?

To produce good systems, we must use 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 professional 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 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 the latter.)

Management and communications

Management experts use the term *high reliability organization* (HRO) for an organization (business or government) that operates in difficult environments, often with complex technology, where failures can have extreme consequences (for example, air traffic control, nuclear power plants).[31] Researchers have identified characteristics of HROs that perform extremely well. These characteristics can improve software and computer systems in both critical and less critical applications. One characteristic is "preoccupation with failure." That means always assuming something unexpected can go wrong—not just planning, designing, and programming for all problems the team can foresee, but always being aware that they might miss something. Preoccupation with failure includes being alert to cues that might indicate an error. It includes fully analyzing near failures (rather than assuming

the system "worked" because it averted an actual failure) and looking for systemic reasons for an error or failure rather than focusing narrowly on the detail that was wrong. (For example, *why* did some programmers for the Mars Climate Orbitor assume measurements were in English units while others assumed metric?)

Another feature of successful organizations is loose structure. It should be easy for a designer or programmer to speak to people in other departments or higher up in the company without going through rigid channels that discourage communication. An atmosphere of open, honest communication within the organization and between a company and client are essential for learning of problems early and reducing the effort required to handle them.

There is much more to the field of organizational features that encourage success. It is well worthwhile for project managers, founders of start-up companies, and anyone in management to devote time to studying it.

Safety-critical applications

A subfield of computer science focuses on design and development of safety-critical software. 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."[32]

To illustrate two important principles in safety-critical applications, I will use as examples accidents that destroyed two space shuttles, each killing the seven people onboard. Computer systems and software were not the cause, but these tragedies make the points well. Burning gases leaked from a rocket shortly after launch of the *Challenger* and destroyed it. The night before the scheduled launch, the engineers argued for a delay. They knew the cold weather posed a severe threat to the shuttle. We cannot prove absolutely that a system is safe, nor can we usually prove absolutely that it will fail and kill someone. An engineer reported that, in the case of the *Challenger*, "It was up to us to prove beyond a shadow of a doubt that it was not safe to [launch]."[33] For the ethical decision maker, the policy should be to suspend or delay use of the system in the absence of a convincing case for safety, rather than to proceed in the absence of a convincing case for disaster. In the second accident, a large piece of insulating foam dislodged and struck the wing of the *Columbia* space shuttle as it launched. NASA knew this happened, but pieces of foam had dislodged and struck the shuttle on other flights without causing a major problem. Thus NASA managers declined to pursue available options to observe and repair the damage. *Columbia* broke up when reentering the earth's atmosphere at the end of its mission. This tragedy illustrates the danger of complacency. An organization focused on safety

must explore ambiguous risks. Tragedies are less likely if the organization has established policies and procedures to evaluate such risks.[34]

Specifications

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. 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.

User interfaces and human factors

If you are editing 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 problems. System designers and programmers need to learn from psychologists and human-factors 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, Colombia, illustrates the importance of consistency (and other aspects of good user interfaces). While approaching 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.[35]

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. One jury assigned some of the responsibility to the companies that provided the computer system. While it is clear that the pilot could have

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

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 the dangerous situation.

Crashing into mountains was a major cause of air travel fatalities. The Cali crash triggered the adoption of a ground proximity warning system (GPWS) to reduce such crashes. Older radar-based systems sometimes gave warning only 10 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 GWPS is likely responsible for preventing crashes in several incidents in which pilots incorrectly set an altimeter, attempted to land with poor visibility, mistook building lights for airport lights, and so on. No commercial U.S. airliner has crashed into a mountain since the GPWS was implemented.[36]

As an illustration of more principles that can help build better and safer systems, we consider several aspects of automated flight systems. An expert in this area emphasizes the following points:[37]

- *The user needs feedback to understand what the system is doing at any time.* This is critical when a 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 an 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.

Redundancy and self-checking

Redundancy and self-checking are two techniques important in systems on which lives and fortunes depend. Redundancy takes several forms. On aircraft, several computers can control an accuator on, say, a wing flap. If one computer fails, another can do the job. Software modules can check their own results—either against a standard or by computing

the same thing in two different ways and then comparing to see if the two results match. A more complex form of redundancy, used, for example, in flight control systems in aircraft, aims to protect against consistently faulty assumptions or methods of one programming team. Three independent teams write modules for the same purpose, in three different programming languages. The modules run on three separate computers. A fourth unit examines the outputs of the three modules and chooses the result obtained by at least two out of three. 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.[38] Thus, this type of "voting" redundancy, while valuable in many safety-critical applications, might not overcome problems in other areas of the software development process.

Testing

It is difficult to overemphasize the importance of adequate, well-designed 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.

A practice called independent verification and validation (IV&V) can be very useful in finding errors in software systems. IV&V means that an independent company (that is, 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. IV&V is helpful for two reasons. The people who designed and/or developed a system think the system works. They think they thought about potential problems and solved them. With the best of intentions, they tend to test for the problems they have already considered. Also, consciously or subconsciously, the people who created the system may be reluctant to find flaws in it. Their testing may be half-hearted. Independent testers bring different perspectives, and for them, success in finding flaws is not emotionally or professionally tied to responsibility for those flaws.

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, the 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.

> *We are what we repeatedly do. Excellence, therefore, is not an act, but a habit.*
>
> —Will Durant, summarizing Aristotle's view in his *Nicomachean Ethics*[39]

8.3.2 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.

Like antilock braking systems in automobiles that control braking to avoid skidding (and do a better job than human drivers), computer systems in airplanes control sudden sharp climbs to avoid stalling. Some airplanes automatically descend if they detect cabin depressurization and the pilot does not take action quickly.

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.[40] The TCAS systems functioned correctly when a Russian airplane carrying many children and a German cargo plane got too close to each other. 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. Pilots of the Airbus 380, the world's largest passenger airplane, 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.[41]

Computers in some airplanes prevent certain actions even if the pilot tries them (for example, 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 otherwise: 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.

8.3.3 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." 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 uphold 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 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 and accuracy of data in databases, 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. (In many contexts, the government does not permit lawsuits against it.)

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 United States for years. The complexity of large computer systems make designing liability standards difficult, but this is a necessary task.

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.[42] 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.[43]

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.[44] Some see a requirement for a government-approved license as a fundamental violation of the freedom to work (that is, of the negative right, or liberty, to work, in the terms of Section 1.4.2).

Clashes between licensing laws and the Web: Section 3.2.5

There are 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 (without a lawsuit). 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 website as low as about $25 for flights between the United States 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 can lead to 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 and provide ways to deal with the risk of failures. Insurance companies have an incentive to evaluate the systems they insure and require that certain standards are met. Some businesses pay a higher rate for "uninterrupted" satellite communications 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 websites that review new programs, or consult one's social network. 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 the seller 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?

Many people who write about the social impacts of computers lament our dependence on computing technology. Because of their usefulness and flexibility, computers, cellphones, and similar devices 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 use of the technology and its gadgets. Is that 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."[45] 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. It did not matter whether her utility bills were paid automatically.

On the other hand, many people and businesses are not prepared to do without the computer systems and electronic devices they use every day. Many drivers would be lost if their navigation system failed. A BlackBerry email blackout disrupted the work of bankers, technology workers, talent agents, and others who depend on constant communication—some who receive more than 500 emails 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 communications services. Drivers could not buy gasoline with their credit cards. "Customers were really angry," said a gas station manager. More than 1000 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."[46]

Is our "dependence" on electronic technology different from our dependence on electricity, which we use for lighting, entertainment, manufacturing, medical treatments—just about everything? Is our "dependence" on computers different from a farmer's dependence on a plow? Modern surgery's dependence on anesthesia?

Computers, smartphones, 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 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. The breakdown can remind us, for example, of the billions of communications, carrying voice, text, photos, and data, that are possible or more convenient or cheaper because of the technology.

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 systems 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. The driver with a navigation system might choose to keep a map in the car.

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 older technologies when we turn on a light or ride a bicycle. 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 United States 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 discover and solve problems. Scientists and engineers study disasters and learn how to prevent them and how to recover from them. A disastrous fire led to the development of fire hydrants—a way to get water to the fire 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.[47] Software engineering textbooks use the Cali crash, described in Section 8.3.1, as an example so that future software specialists will not repeat the mistakes in the plane's computer system. We learn. Overall, computer systems and other technologies have made air travel safer. In the first decade of this century, there was roughly one fatal accident per four million commercial flights, down 60% from 10 years earlier.[48]

The death rate from motor vehicle accidents in the United States declined almost 80% from 1965 to 2010 (from 5.30 per 100 million vehicle miles traveled to 1.13 per 100 million vehicle miles traveled).[49] Why? One significant factor is increased education about responsible use (i.e., the campaign against drunk driving). Devices that protect people when the system fails (seat belts and airbags) are another. Other systems help avoid accidents: Rear-view cameras help drivers avoid hitting a child when backing up. "Night vision" systems detect obstacles and project onto the windshield an image or diagram of objects in the car's path. Electronic stability 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 5 in 100,000 in 2008.[50]

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, workers and planners in other fields, and as members of the public.

Observations

Throughout this chapter, 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. There is a "learning curve" for new technologies. By studying failures, we can reduce their occurrence.
3. Much is known about how to design, develop, and use complex systems well and safely. Ethical professionals learn and follow these methods.
4. Perfection is not an option. The complexity of computer systems makes errors, oversights, and failures likely.
5. Comparing the risks of using computer technologies with the risks of using other methods, and weighing the risks against the benefits, give us important perspective.

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 a particular application area or when evaluating the technology as a whole that we should look at the balance between risks and benefits.

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