**Chapter 8: Errors, Failures, And Risks**

8.1: Failures and Errors in Computer Systems

8.1.1: An Overview

\*Navigation System Directs Car Into River

\*Flaw Found in Software that Tracks Nuclear Materials

\*Robot Kills Worker

\*Man arrested Five Times due to faulty FBI Computer Data

Most computer applications are so complex that it is virtually impossible to produce programs with no errors.

We should understand computer related problems from the perspective of several roles we play:

\*A computer user: we should understand the limitation of computers and the need for proper training and responsible use. We must recognize that, as in other areas, there are good products and bad products.

\*A computer professional: Studying computer failures should help you become a better computer professional. Understanding the source and consequences of computer failures is also valuable if you will be responsible for buying, developing, or managing a complex system for a hospital, airport or business.

\*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.

8.1.2 Problems for Individuals

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

**Billing Errors**

The first few errors we look at are relatively simple ones whose negative consequences:

* 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. The computer generated erroneous for almost 5,000 people. One woman receives a tax bill for $40,000,000,541.13.

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 error, good systems have provisions for checking their results.

**Inaccurate and misinterpreted data in database**

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. Many people battle for years to get the credit bureaus to correct information in their records, and a few have won large settlements in lawsuits. Federal law requires states to maintain databases of people convicted of sex crimes against children and to release information about them to the public.

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.

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

* A large population (many people have identical or similar names, and most of our interactions are with strangers)
* Automated processing without human common sense or the power to recognize special cases
* Automated processing without human common sense or the power to recognize special cases
* Overconfidence in the accuracy of data stored on computers
* Errors (some because of carelessness) in data entry.
* Failure to update information and correct errors
* Lack of accountability for errors

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

Blackberry thumb and RSI:

Millions of children use small electronic devices and adults answer e-mail on portable electronic devices with mini keypads. This issue or injury (RSI) covers a variety of injuries or pain in thumb, fingers, wrists and arms.

8.1.3 System failures

Modern communication, power, medical, financial, retail, and transportation systems depend heavily on computer systems. They do not always function as planned.

**Communication, business, and transportation**

Customers of AT&T lost telephone service for voice and data for nine hours because of a software error in a four million-line program. The disruption prevented roughly 50 million calls from getting through.

Every few years, the computer system of one of the world’s largest stock exchanges or brokerages fails. An error in software upgrade shutdown trading on Tokyo Stock Exchange. A problem in communications software virtually shutdown the NASDAQ stock exchange for two and a half hours.

**Destroying businesses**

Several companies have gone bankrupt after spending a huge amount of money on computer systems that failed to work.

A few dozen companies that bought an inventory system called Warehouse Manager blamed the system for disastrous losses. One previously successful company successful saw its income decline by about half and laid off half its employees.

What was responsible for the problems in Warehouse Manager? NCR Corporation sold the program, but another company developed it. The company originally designed and implemented the program on a different computer and operating system. It appears that they were unexpected problems when the company rewrote the program for NCR’s machines and its ITX operating system. Memos at NCR reported inadequate testing and poor performance in real business settings.

CTB/McGraw-Hill develops and scores standardized tests for schools. An error in CTB’s software caused it to report test result incorrectly-substantially lower than the correct scores. In New York, school principals lost their jobs because their schools appeared to be doing a poor job of teaching students to read. Nearly 9,000 students had to attend summer school because of the incorrect scores. Eventually, CTB correct the error. New York City’s reading scores had actually risen five percentage points.

**Voting System**

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.

A programming error generated 100,000 extra votes in a Texas county. A programming error caused some candidates to receive votes actually cast for other candidates.

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.

One policy that can help reduce errors and fraud it to make the software in electronic voting machines public. Then various experts and tech-savy members of the public can examine it. Companies that produce electronic voting systems have resisted this, arguing that the software is a propriatery. If they release it, they may lose an advantage over competitors.

🡪Electronic systems have the potential for reducing some kinds of fraud and accidental loss of ballots but they have not yet reached the level of security to ensure a reasonable degree of trust.

**Stalled airports: Denver, Hong Kong, and Malaysia**

Denver International Airport, a $3.2 billion airport. The opening was rescheduled at least four times. The delay cost more than $30 million per month. The computer controlled baggage-handling system, which cost $193 million, caused most of the delay. The system should have worked as follow: inbound luggage would go to terminals or transfer directly to connecting flights anywhere in the airport. The system did not work as planned.

Some of the specific problems were:

* Real-world problems: some scanners got dirty or knocked out of alignment and could not detect carts going by.
* Problems in other systems: the airport’s electrical system could not handle the power surges 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.

What led to the extraordinary delay in the Denver baggage system were two main causes:

\* The time allowed for development and testing of the system was insufficient: it had to be finished only in two years, while the one at Frankfurt Airport in Germany spent six years in development and two years testing.

\* Denver made significant changes in specifications after the project began. The system as only done 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 baggage system installed at San Francisco International Airport.

🡪 Bottom line lesson is that system designers must build in plenty of testing and debugging time when scaling up proven technology into a much more complicated environment.

**Abandoned Systems**

The flaws in many systems are so fundamentals 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.

Many projects require much more time and money than originally planned. Some are never completed because they are beyond the capabilities of the current technology. Software expert Robert Charette estimates that from 5% to 15% of information technology projects are abandoned before or soon after delivery – out of about $ 1 trillion spent each year worldwide.

**Legal Systems**

After US Airways and America West merged, they combined their reservation system. The system failed. Long lines of tickets counters delayed thousands of passengers. Merging different computer systems is extremely tricky. Some of the reported problems were out of date systems (hardware, software or peripheral equipment).

The problems of legacy systems are numerous. Old hardware fails, and replacement parts are hard to find. Old software often runs on new hardware, but it is still old software. Programmers no longer learn the old programming languages. The programmers who wrote the software operated the systems have left the company, retired or dead. Old programs often had little or no documentation. If there were good design documents and manuals, they probably no longer exist or cannot be found.

🡪Some High Level causes of Computer System Failures:

* Lack of clear, well-though-out goals and specifications.
* Poor management and poor communication among customers, designers, programmers, and so on.
* Institutional or political pressures that encourage unreaslistically 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.

The Y2K Problem:

Older software, designed for computers with very limited storage space, typically used two digits to represent the year.

Many computer experienced problems using dates in or after the year 2000.

For example, some credit cards with expiration dates in 2000 would not work. The software interpreted the expiration date as 1900. Software to calculate payments on loans due after 2000 failed.

Business and governments spend many billions of dollars on the Y2K problems, tracking down two-digit dated in their software and making modifications, rushing to finish before January 1, 2000. Programmers took special courses in old programming languages (such as COBOL).

As January 1, 2000 approached, many people feared major disasters but nothing happened. Afterward, some argued that the huge multiyear, multibillion-dollar effort to upgrade the system and fix the software succeeded.

**8.1.4 Safety Critical Applications**

**Computers in the air**

The A320 Airbus airplane was the first fully “fly-by-wire” airplane. Pilots do not directly control the plane. Their actions are inputs to computers that control the aircraft systems. Between 1988 and 1993, four A320s crashed. Although investigators decided that some of the crashes were the pilot error, pilots blamed the fly-by-wire system. Pilots complained that the airplane does not respond as expected. The official report on the crash indicated that reasons for the error probably included pilots’ lack of familiarity with the automation equipment and confusing design of the control and displays. Perhaps pilots had too much confidence when the error happened that the computer ability will detect and fix the error.

**8.1.4 Perspective on Failure**

How close to perfection should we expect billing systems to be? A water-utility company sent a customer an incorrect bill for $22,000. A spokesman for the company pointed that one incorrect bill out of 275,000 monthly bills is a good error rate. With approximately eight billion ATM transactions each year, there is one error in rough 45,000.

How accurate should software for check processing be? 99% or 99.9%?

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

**Trust the human or the computer system?**

How much control should computers have in a crisis?

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 system had many flaws but the newer system was heavily improved. It is now a great advance in safety. The systems detected a potential collision and told the Russian pilot to climb and the German pilot to descend. US and European pilots are now trained to follow TCAS instructions EVEN IF THEY CONFLICT WITH INSTURCTIONS FROM AN AIR TRAFFIC CONTROLLER.

Some airlines disable parts of computer systems in their planes because they do not trust the plane. However, there are circumstances where the computer can do better than most people.

The computers in some airplanes prevent certain actions even if the pilot tries them. Some people object, arguing that the pilot should have ultimate control in case unusual action is needed in emergency.

**8.2 Case Study: The Therac-25**

8.2.1 The Therac-25 Radiation Overdoses

The benefits of computing technology to health care are numerous and very impressive.

The Therac-25 was a software controlled radiation therapy machine used to treat people with cancer. Between 1985 and 1987, Therac-25 machines at four medical centers gave massive overdoses of radiation to six patients. In some cases, the operator repeated an overdose because the machine’s display indicated that it had given no dose. Medical personnel later estimated that some patients received between 13,000 and 25,000 radiations, and painful injuries and the deaths of three patients.

Studies in 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.

(The Therac-25 can generate an electron beam or an X-ray photon beam. The type of beam needed depends on the tumor being treated. It is essential that the proper protective device be in place when the electron beam is on. A third position of the turntables 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 software was apparently assumed to be functioning correctly before.

The Therac-25 malfunctioned frequently. One facility said there were sometimes 40 does rate malfunctions in a day, generally under doses. Thus, operators became used to error message appearing often, with no indication that there might be safety hazards. The error message appeared on the display were simply error number of obscure messages (“Malfunction 54” or “H-tilt”). One had to look up each error number in a manual for more explanation. The operator’s manual for the Therac-25, however, did not include any explanation of the error messages. The maintenance manual did not explain them earlier.

Bugs

Investigators were able to trace some of the overdoses to two specific software errors.

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 the device was checked, each time Set-Up Test procedure ran, it incremented the variable to make it nonzero. The problem was that the flag variable stored in one byte. After the 256th call to the routine, the flag overflowed and showed a value of zero.

It was a simple mistake that has a simple correction. The solution is to set the flag variable to a fixed value 1, rather than incrementing it, to indicate that the device needs checking.

8.2.3 Why So Many Incidents?

 There were 6 knows 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 2 years at some clinics. They did not immediately pull it from service after the first few accidents because they did not know immediately that it caused the injuries. After the second accident, the AECL investigated and found several problems related to the turntable. They eventually installed the critical hardware safety interlocks, and most of the machines remained in use with no incidents of overdoses after 1987.

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. In one case where a clinic added its own hardware safety features to the machine, the AECL told them that it was unnecessary (none of the accidents occur at that facility). For example, operators ignored error messages because the machine produced so many of them.

8.2.4 Observations and Perspective

The manufacturer of the Therac-25 did a poor job. 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, 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.

**8.3 Increasing Reliability and Safety**

8.3.1 What Goes Wrong?

Computer systems fail for two general reasons: The job they are doing is inherently difficult, and the job is often done poorly.

Computer systems now interact with the real world and include complex communications networks, have numerous features and interconnected subsystems, and are extremely large.

**Overconfidence**

Overconfidence, or an unrealistic understanding of the risks in a complex computer system, is a core issue.

**! Some Factors in Computer System Errors and Failures:**

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

**Reuse of Software: The Ariane 5 rocket and “No Fly” lists**

Less than 40 seconds after the launch of the first Ariane 5 rocket, the rocket veered and off course it was destroy as a safety precaution. The rocket and the satellites it was carrying cost approximately $500 million. A software error caused the failure. 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 lunch on the 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) causing the system to halt.

**Failure to update**

Failure to update information in databases has led to problems of varying severity.

The FBI maintains a database that contains criminal history records from participating states and is available to police throughout most of the country. Privacy Journal reported that about half of the records do not indicate whether a suspect was convicted or exonerated.

8.3.2 Professional Techniques

**Software Engineering and professional responsibility**

The many examples of computer systems errors and failures suggest the importance of using good software engineering techniques at all stages of development, including specifications, design, implementation, documentation, and testing.

Professionals both programmers and managers, have the responsibility to study and use the techniques and tools that are available and follow the procedures and guidelines established in the various relevant code of ethics and professional practices (The ACM/IEEE-CS Software Engineering Code of Ethics and Professional Practice and the ACM Code of Ethics and Professional Conduct).

Mistakes will happen, and unexpected problems will arise. An atmosphere of open, honest communication within the software problem will arise. An atmosphere of open, honest communication within the software development company and between the company and the client is essential for learning of problems at an early stage.

**User interfaces and human factors**

Well-designed user interfaces can help avoid many computer related problems. System designers and programmers need to learn from psychologists and human-factor experts who know principle and practices for doing a good job. User interfaces should provide clear instructions and error messages.

As an illustration of more principles that can help build safer systems, we should do the following:

* The pilot needs feedback to understand what the automated system is doing at anytime.
* The system should behave as the pilot expects
* A workload what is too law 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

Good user interfaces are essential in safety critical applications.

**Redundancy and self-checking**

Redundancy and self checking are two important techniques in system on which lives and fortunes depend. The space shuttle in 1980s used four identical but independent computer systems that received input from multiple sensors and checked whether their results against each other. If one computer disagreed with the other three, it was taken out of service.

**Testing**

It is difficult to overemphasize the importance of adequate well planned testing of software. Testing is not arbitrary. There are principles and techniques for doing a good job. Many significant computer system failures occur after installation of an update or upgrade. Unfortunately, many cost conscious managers, programmers and software developers see testing as a step you can skip to save money and time.

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 chargers when fraud or criminal negligence occurs. Families of Therac-25 victims sued.

Many contracts for business computer systems limit the amount the customer can recover to the account amount spent on the computer system. Courts have upheld such contract limitations.

**Warranties for consumer software**

Most mass, retail consumer software come with licensing agreements that indicate you are buying the software “as-is”. There is no guarantee that it works correctly. And such agreements also include provisions prohibiting the user from publically criticizing. Consumer advocates disagree on the extent to which the law should uphold such agreements. As some of them argue for mandatory warranties on software, for making software companies pay for bugs.

**Regulation and safety critical application**

Is there legislation or regulation that can prevent life-threatening computer failures?

A widely accepted option is regulation, possibly including specific testing requirement for approval by a government agency before a new product can be sold. Companies must do expensive testing, provide huge quantities of documentation, and get government approval before they sell new drugs and some medical devices. 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.

**Professional licensing**

Another controversial approach to improving software quality is mandatory licensing of software development professionals. 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.

**Taking responsibility**

In some cases of computer errors, business pay customers for problems or damages.

Insurance companies have an incentive to evaluate the systems they insure.

**8.4 Dependence, Risk and Progress**

8.4.1 Are we too dependent on computers?

Many people and businesses are not prepared to do without the computer systems the use everyday. A BlackBerry email blackout lasted nine hours. It disrupted the work of bankers, technology workers, talent agents and other who depend on such technology.

8.4.2 Risk and Progress

We trust our lives to technology everyday. We trust older, non-computer technologies every time we step into an elevator, a car or a building. As the tools and technologies we use become more complex and more interconnected, the amount of damage that results from an individual disruption or failure increases, and sometimes we pay the costs in dramatic and tragic events.

As use of technology, automation, and computer systems has increased virtually all work places, the risk of dying dropped from 39 among 100,000 workers in 1934 to four in 100,000 in 2004. However sometimes computer technology failures has major negative consecutive than other non-computer technologies, and the change in computer systems is very fast.

**Observations**

We have several points

1. Many of the issues related to reliability and safety for computer systems have arisen before with other technologies
2. Perfection is not an option. The complexity of computer systems makes errors.
3. There is a learning curve for new technologies.
4. We should compare risks of using computers with other methods.

This does not mean we should excuse or ignore computer errors because failures occur. It does not mean we should tolerate carelessness.

**Chapter 9**

**9.3.5 Skipping Tests**

There are often pressures for reducing testing of software. Testing is one of the last steps in development, so when deadlines approach, testing schedules often shrink.