Overview

- Software disasters happen many ways:
  - Data-entry or data-retrieval errors
  - Software and billing errors
  - Poor Programming Practices
  - Attacks
- Notable software system failures
  - Therac-25
  - Boeing Max-8
  - Toyota Brakes
  - VW Diesel Emissions
- Computer simulations
- Software engineering
- Software warranties

Introduction

- Computer systems are sometimes unreliable
  - Erroneous information in databases
  - Misinterpretation of database information
  - Malfunction of embedded systems
- Effects of computer errors
  - Inconvenience
  - Bad business decisions
  - Fatalities

Software Reliability Cases

Optimistic people would like to believe that once the software performs correctly, it will be correct forever.

Tragedies in Therac-25, a computer-controlled radiation-therapy machine in the year 1986, caused by the software not being able to detect a race condition, alerts us that it is dangerous to abandon our old but well-understood mechanical safety control and surrender our lives completely to software controlled safety mechanism.

Software can make decisions, but can just as unreliable as human beings.

- The British destroyer Sheffield was sunk because the radar system identified an incoming missile as “friendly”.
- The defense system has matured to the point that it will not mistake the rising moon for incoming missiles, but gas-field fires, descending space junk, etc, were also examples that can be misidentified as incoming missiles by the defense system.

Software Reliability Cases

Software can also have small unnoticeable errors or drifts that can culminate in a disaster.

- On February 25, 1991, during the Golf War, the chopping error that missed 0.000000095 second in precision in every 10th of a second, accumulating for 100 hours, made the Patriot missile fail to intercept a scud missile.
- 28 lives were lost.

Fixing problems may not necessarily make the software more reliable.

- On the contrary, new serious problems may arise.
- In 1991, after changing three lines of code in a signaling program which contains millions lines of code, the local telephone systems in California and along the Eastern seaboard stopped working.

Once perfectly working software may also break if the run environment changes.

- After the success of Ariane 4 rocket, the maiden flight of Ariane 5 ended up in flames while design defects in the control software were unveiled due to the faster horizontal drift speed of the new rocket.

Two Kinds of Data-related Failure

- A computerized system may fail because wrong data was entered into it
- A computerized system may fail because people incorrectly interpret data they retrieve
Disfranchised Voters

- November 2000 general election
- Florida disqualified thousands of voters
- Reason: People identified as felons
- Cause: Incorrect records in voter database
- Consequence: May have affected election’s outcome

False Arrests

- Sheila Jackson Stossier mistaken for Shirley Jackson
  - Arrested and spent five days in detention
- Roberto Hernandez mistaken for another Roberto Hernandez
  - Arrested twice and spent 12 days in jail
- Terry Dean Rogan arrested after someone stole his identity
  - Arrested five times, three times at gun point

Accuracy of NCIC Records

- March 2003: Justice Dept. announces FBI not responsible for accuracy of NCIC information
- Exempts NCIC from some provisions of Privacy Act of 1974
- Should government take responsibility for data correctness?

Dept. of Justice Position

- Impractical for FBI to be responsible for data’s accuracy
- Much information provided by other law enforcement and intelligence agencies
- Agents should be able to use discretion
- If provisions of Privacy Act strictly followed, much less information would be in NCIC
- Result: fewer arrests

Position of Privacy Advocates

- Number of records is increasing
- More erroneous records → more false arrests
- Accuracy of NCIC records more important than ever

Analysis: Database of Stolen Vehicles

- > 1 million cars stolen every year
  - Owners suffer emotional, financial harm
  - Raises insurance rates for all
- Transporting stolen car across a state line
  - Before NCIC, greatly reduced chance of recovery
  - After NCIC, nationwide stolen car retrieval
- At least 50,000 recoveries annually due to NCIC
- Few stories of faulty information causing false arrests
- Benefit > harm → Creating database the right action
Errors When Data Are Correct

- Assume data correctly fed into computerized system
- System may still fail if there is an error in programming

Errors Leading to System Failures

- Ambulance dispatch system in London
- Chicago Board of Trade
- BMW limousine
- Japan's air traffic control system
- Los Angeles County + USC Medical Center laboratory computer system
- Comair's Christmas Day shutdown
- Boeing 777

Errors Leading to System Malfunctions

- Qwest sent incorrect bills to cell phone customers
- Faulty USDA beef price reports
- U.S. Postal Service returned mail addressed to Patent and Trademark Office
- Spelling and grammar error checkers increase errors
- New York City Housing authority overcharged renters
- About 450 California prison inmates mistakenly released

Errors Leading to System Failures

- Analysis: E-Retailer Posts Wrong Price, Refuses to Deliver
  - Amazon.com in Britain offered iPaq for £7 instead of £275
  - Orders flooded in
  - Amazon.com shut down site, refused to deliver unless customers paid true price
  - Was Amazon.com wrong to refuse to fill the orders?

Rule Utilitarian Analysis

- Imagine rule: A company must always honor the advertised price
- Consequences
  - More time spent proofreading advertisements
  - Companies would take out insurance policies
  - Higher costs → higher prices
  - All consumers would pay higher prices
  - Few customers would benefit from errors
- Conclusion
  - Rule has more harms than benefits
  - Amazon.com did the right thing

Kantian Analysis

- Buyers knew 97.5% markdown was an error
- They attempted to take advantage of Amazon.com's stockholders
- They were not acting in "good faith"
- Buyers did something wrong
**Patriot Missile**

- Designed as anti-aircraft missile
- Used in 1991 Gulf War to intercept Scud missiles
- One battery failed to shoot at Scud that killed 28 soldiers
- Designed to operate only a few hours at a time
- Kept in operation > 100 hours
- Tiny truncation errors added up
- Clock error of 0.3433 seconds → tracking error of 687 meters
- Prior to the incident, Army officials had fixed the software to improve the Patriot systems accuracy.
  - That modified software reached the base the day after the attack.

**Pentium FDIV bug**

When a math professor discovered and publicized a flaw in Intel's popular Pentium processor in 1994, the company's response was to replace chips upon request to users who could prove they were affected.

Intel calculated that the error caused by the flaw would happen so rarely that the vast majority of users wouldn't notice.

Angry customers demanded a replacement for anyone who asked, and Intel agreed.

The episode cost Intel $475 million.

**Knight's $440 Million Error**

One of the biggest American market makers for stocks struggled to stay afloat after a software bug triggered a $440 million loss in just 30 minutes.

The firm’s shares lost 75 percent in two days after the faulty software flooded the market with unintended trades.

Knight’s trading algorithms reportedly started pushing erratic trades through on nearly 150 different stocks, sending them into spasms.

**Boeing 737-Max 8**

The culprit in the Max-8 crashes was the Maneuvering Characteristics Augmentation System (MCAS).

Like the 737 Max, MCAS was made to be a stopgap.

The Max was designed around a new set of engines called LEAP-1Bs. These are much more efficient than the engines on the 737NG, but they are also much heavier and larger.

This created a design problem. The engines on the NG sit only 18 inches off the ground, and mounting the LEAP-1Bs in the same spot gave them too little clearance during takeoff. So Boeing placed them further forward and slightly higher on the wing of the Max.

That solution created an aerodynamics problem. Due to their size and position, the engines on the Max create lift when the airplane enters a steep climb (or, in aviation parlance, at high angles of attack). This extra lift causes the Max to handle differently than previous versions of the 737, but only when it's climbing steeply.

**Boeing Orion Starliner**

The Starliner test flight, which was intended to test the automated systems and did not carry any astronauts, ended up in the wrong orbit. The mission was cut short, without docking at the International Space Station and likely delaying plans that are already a couple of years behind schedule.

The first test of Starliner, built by Boeing's space and defense division, was postponed multiple times this year.

The Atlas 5 performed as designed, placing the capsule on an elliptical trajectory.

The capsule itself was to make a final maneuver that would shift the orbit from elliptical to circular and allow it to meet up with the International Space Station on Saturday.
**Boeing Orion Starliner**

The spacecraft’s clock was set to the wrong time, and a flawed thruster burn pushed the capsule into the wrong orbit.

- Its internal clock was off by 11 hours, which led it to miss the key moment to fire its engines.

“We don’t understand the root cause,” said Jim Chilton, senior vice president of the space and launch division of defense, space and security segment of Boeing.

Attempts to send a command to fix the problem apparently did not reach the spacecraft because it was in between satellite communication links, and it used too much propellant to continue to the space station.

A second timer error was found during the flight. The timer error on Starliner was discovered by ground controllers during the flight and corrected while the spacecraft was still aloft, but panel member Paul Hill said the consequences of the error would have been grave if it had gone unnoticed.

“While this anomaly was corrected in flight, if it had gone uncorrected, it would have led to erroneous thruster firings and uncontrolled motion during [service module] separation for deorbit, with the potential for a catastrophic spacecraft failure,” Hill said in the meeting.

**NOAA-19 Satellite**

Although not a software error, on September 6, 2003, this satellite was badly damaged while being worked on at the Lockheed Martin Space Systems factory.

- The satellite fell to the floor as a team was turning it to a horizontal position.
- An inquiry into the mishap determined that it was caused by a lack of procedural discipline throughout the facility.
- Turns out that while the turn-over cart used during the procedure was in storage, a technician removed twenty-four bolts securing an adapter plate to it without documenting the action.
- The team subsequently using the cart to turn the satellite failed to check the bolts, as specified in the procedure, before attempting to move the satellite.

Repairs to the satellite cost $135 million.

**Patriot Missile Failure**

![Diagram of Patriot missile failure](image)

**Ariane 5**

- Satellite launch vehicle
- 36 seconds into maiden flight, rocket self-destructed
- $500 million of uninsured satellites lost
- Statement assigning floating-point value to integer raised exception
  - Software had tried to cram a 64-bit number into a 16-bit space.
- Exception not caught and computer(s) crashed
- Code reused from Ariane 4
  - Slower rocket
  - Smaller values being manipulated
  - Exception was impossible

*$8 Billion development cost

**EDS Child Support System**

In 2004, EDS introduced a highly complex IT system to the U.K.’s Child Support Agency (CSA).

At the exact same time, the Department for Work and Pensions (DWP) decided to restructure the entire agency.

The two pieces of software were completely incompatible, and irreversible errors were introduced as a result.

The system somehow managed to overpay 1.9 million people, underpay another 700,000, had US$7 billion in uncollected child support payments, a backlog of 239,000 cases, 36,000 new cases “stuck” in the system, and has cost the UK taxpayers over US$1 billion to date.
**Soviet Gas Pipeline Explosion**

The Soviet pipeline had a level of complexity that would require advanced automated control software. The CIA was tipped off to the Soviet intentions to steal the control system’s plans.

Working with the Canadian firm that designed the pipeline control software, the CIA had the designers deliberately create flaws in the programming so that the Soviets would receive a compromised program.

It is claimed that in June 1982, flaws in the stolen software led to a massive explosion along part of the pipeline, causing the largest non-nuclear explosion in the planet’s history.

---

**Bitcoin Hack, Mt. Gox**

Launched in 2010, Japanese bitcoin exchange, Mt. Gox, was the largest in the world.

After being hacked in June, 2011, Mt. Gox stated that they’d lost over 850,000 bitcoins (worth around half a billion US dollars at the time of writing).

Although around 200,000 of the bitcoins were recovered, Mark Karpeles admits “We had weaknesses in our system, and our bitcoins vanished.”

---

**Heathrow Terminal 5 Opening**

Just before the opening of Heathrow’s Terminal 5 in the UK, staff tested the brand new baggage handling system built to carry the vast amounts of luggage checked in each day.

Engineers tested the system thoroughly before opening the Terminal to the public with over 12,000 test pieces of luggage.

It worked flawlessly on all test runs only to find on the Terminal’s opening day the system simply could not cope.

It is thought that “real life” scenarios such as removing a bag from the system manually when a passenger had left an important item in their luggage, had caused the entire system to become confused and shut down.

Over the following 10 days some 42,000 bags failed to travel with their owners, and over 500 flights were cancelled.

---

**AT&T Long-Distance Network**

- **Significant service disruption**
  - About half of telephone-routing switches crashed
  - 70 million calls not put through
  - 60,000 people lost all service
  - AT&T lost revenue and credibility
- **Cause**
  - Single line of code in error-recovery procedure
  - Most switches running same software
  - Crashes propagated through switching network

---

**AT&T Long Distance Network Failure**

- **Mars Climate Orbiter**
  - Disintegrated in Martian atmosphere
  - Lockheed Martin design used English units
  - Jet Propulsion Lab design used metric units
  - $125 million craft hurtling on through space in an orbit around the sun
- **Mars Polar Lander**
  - Crashed into Martian surface
  - Engines shut off too soon
  - False signal from landing gear

---

**Robot Missions to Mars**

- **Mars Climate Orbiter**
- **Mars Polar Lander**
More Nasa Problems

The Mariner 1 Spacecraft

On a mission to fly-by Venus in 1962, this spacecraft barely made it out of Cape Canaveral when a software-coding error caused the rocket to veer dangerously off-course, threatening to crash back to earth. Alarmed, NASA engineers on the ground issued a self-destruct command. A review board later determined that the omission of a hyphen in coded computer instructions allowed the transmission of incorrect guidance signals to the spacecraft. The cost for the rocket was reportedly more than $18 million at the time.

The Morris Worm

A program developed by a Cornell University student for what he said was supposed to be a harmless experiment wound up spreading wildly and crashing thousands of computers in 1988 because of a coding error. It was the first widespread worm attack on the fledgling Internet. The graduate student, Robert Tappan Morris, was convicted of a criminal hacking offense and fined $10,000. Morris’s lawyer claimed at the trial that his client’s program helped improve computer security. Costs for cleaning up the mess may have gone as high as $100 Million. Morris, who interestingly co-founded the startup incubator Y Combinator, is now a professor at the Massachusetts Institute of Technology. A disk with the worm’s source code is now housed at the University of Boston.

Denver International Airport

- BAE built automated baggage handling system
- Problems
  - Airport designed before automated system chosen
  - Timeline too short
  - System complexity exceeded development team’s ability
- Results
  - Added conventional baggage system
  - 16-month delay in opening airport
  - Cost Denver $1 million a day

Tokyo Stock Exchange

- First day of trading for J-Com
- Mizuho Securities employee mistakenly entered order to sell 610,00 shares at 1 yen, instead of 1 share at 610,000 yen
- Employee overrides computer warning
- After sell order posted on exchange’s display board, Mizuho tried to cancel order several times; software bug caused attempts to fail
- Mizuho lost $225 million buying back shares

Direct Recording Electronic Voting Machines

- After problems with 2000 election, Congress passed Help America Vote Act of 2002
- HAVA provided money to states to replace punch card voting systems
- Many states used HAVA funds to purchase direct recording electronic (DRE) voting machines
- Brazil and India have run national elections using DRE voting machines exclusively
- In November 2006 1/3 of U.S. voters used DRE voting machines

Issues with DRE Voting Machines

- Voting irregularities
  - Failure to record votes
  - Overcounting votes
  - Misrecording votes
- Lack of a paper audit trail
- Vulnerability to tampering
- Source code a trade secret, can’t be examined
- Possibility of widespread fraud through malicious programming
Genesis of the Therac-25

Atomic Energy of Canada Limited (AECL) and CGR built Therac-6 and Therac-20.
- These models were manually operated – no computer control.

Therac-25 built by AECL
- PDP-11 computer was an integral part of the system
- Hardware safety features replaced with software
- Reused code from Therac-6 and Therac-20

First Therac-25 shipped in 1983
- Patient in one room
- Technician in adjoining room

Because of concurrent programming errors, it sometimes gave its patients radiation doses that were hundreds of times greater than normal, resulting in death or serious injury.

Chronology of Accidents and AECL Responses

- Marietta, Georgia (June 1985)
- Hamilton, Ontario (July 1985)
- First AECL investigation (July-Sept. 1985)
- Yakima, Washington (December 1985)
- Tyler, Texas (March 1986)
- Second AECL investigation (March 1986)
- Tyler, Texas (April 1986)
- Yakima, Washington (January 1987)
- FDA declares Therac-25 defective (February 1987)

These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics and software engineering.

Additionally the overconfidence of the engineers and lack of proper due diligence to resolve reported software bugs are highlighted as an extreme case where the engineers’ overconfidence in their initial work and failure to believe the end users’ claims caused drastic repercussions.

Post Mortem Analysis

The six documented accidents occurred when the high-current electron beam generated in X-ray mode was delivered directly to patients.

Two software faults were to blame.
- One, when the operator incorrectly selected X-ray mode before quickly changing to electron mode, which allowed the electron beam to be set for X-ray mode without the X-ray target being in place.
- A second fault allowed the electron beam to activate during field-light mode, during which no beam scanner was active or target was in place.

Previous models had hardware interlocks to prevent such faults, but the Therac-25 had removed them, depending instead on software checks for safety.

Post Mortem Analysis

A commission attributed the primary cause to general poor software design and development practices rather than single-out specific coding errors.

In particular, the software was designed so that it was realistically impossible to test it in a clean automated way.

Researchers who investigated the accidents found several contributing causes. These included the following institutional causes:
- AECL did not have the software code independently reviewed.
- AECL did not consider the design of the software during its assessment of how the machine might produce the desired results and what failure modes existed.
- The system noticed that something was wrong and halted the X-ray beam, but merely displayed the word “MALFUNCTION” followed by a number from 1 to 64.
- AECL personnel, as well as machine operators, initially did not believe complaints. This was likely due to overconfidence.
- AECL had never tested the Therac-25 with the combination of software and hardware until it was assembled at the hospital.

Post Mortem Analysis

The researchers also found several engineering issues:
- The failure occurred only when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled the PDP-11 computer.
- The design did not have any hardware interlocks to prevent the electron-beam from operating in its high-energy mode without the target in place.
- The engineer had reused software from older models. Such methods manifest in so-called cargo cult programming where there is blind reliance on previously created code that is poorly understood and may or may not be applicable.
- The hardware provided no way for the software to verify that sensors were working correctly.
- The equipment control task did not properly synchronize with the operator interface task, so that race conditions occurred if the operator changed the setup too quickly.
- The software set a flag variable by incrementing it, rather than by setting it to a fixed non-zero value.
- Occasionally an arithmetic overflow occurred, causing the flag to return to zero and the software to bypass safety checks.
Post Mortem Analysis

• Race condition: order in which two or more concurrent tasks access a shared variable can affect program’s behavior
• Two race conditions in Therac-25 software
  – Command screen editing
  – Movement of electron beam gun

Post Mortem

• AECL focused on fixing individual bugs
• System not designed to be fail-safe
• No devices to report overdoses
• Software lessons
  – Difficult to debug programs with concurrent tasks
  – Design must be as simple as possible
  – Documentation crucial
  – Code reuse does not always lead to higher quality
• AECL did not communicate fully with customers

Moral Responsibility of the Therac-25 Team

• Conditions for moral responsibility
  – Causal condition: actions (or inactions) caused the harm
  – Mental condition
    * Actions (or inactions) intended or willed -OR-
    * Moral agent is careless, reckless, or negligent
• Therac-25 team morally responsible
  – They constructed the device that caused the harm
  – They were negligent

Postcript

• Computer errors related to radiation machines continue to maim and kill patients
• Investigation by The New York Times
  – May 2000: Meet Halfa, Egypt; two fatalities due to radiography accident.

Uses of Simulations

• Simulations replace physical experiments
  – Experiment too expensive or time-consuming
  – Experiment unethical
  – Experiment impossible
• Model past events
• Understand world around us
• Predict the future

Validating Simulations

• Verification: Does program correctly implement model?
• Validation: Does the model accurately represent the real system?
• Validation methods
  – Make prediction, wait to see if it comes true
  – Predict the present from old data
  – Test credibility with experts and decision makers
Specification

- Determine system requirements
- Understand constraints
- Determine feasibility
- End products
  - High-level statement of requirements
  - Mock-up of user interface
  - Low-level requirements statement

Development

- Create high-level design
- Discover and resolve mistakes, omissions in specification
- CASE tools to support design process
- Object-oriented systems have advantages
- After detailed design, actual programs written
- Result: working software system

Validation (Testing)

- Ensure software satisfies specification
- Ensure software meets user’s needs
- Challenges to testing software
  - Noncontinuous responses to changes in input
  - Exhaustive testing impossible
  - Testing reveals bugs, but cannot prove none exist
- Test modules, then subsystems, then system

Software Quality Is Improving

- Standish Group tracks IT projects
- Situation in 1994
  - 1/3 projects cancelled before completion
  - 1/2 projects had time and/or cost overruns
  - 1/6 projects completed on time / on budget
- Situation in 2006
  - 1/6 projects cancelled
  - 1/2 projects had time and/or cost overruns
  - 1/3 projects completed on time / on budget

Success of IT Projects Over Time

Software Failure Analysis
Hardware Life Cycle

Software Life Cycle

Shrinkwrap Warranties

- Some say you accept software “as is”
- Some offer 90-day replacement or money-back guarantee
- None accept liability for harm caused by use of software

Are Software Warranties Enforceable?

- Article 2 of Uniform Commercial Code
- Magnuson-Moss Warranty Act
- Step-Saver Data Systems v. Wyse Technology and The Software Link
- ProCD, Inc. v. Zeidenberg
- Mortensen v. Timberline Software

Moral Responsibility of Software Manufacturers

- If vendors were responsible for harmful consequences of defects
  - Companies would test software more
  - They would purchase liability insurance
  - Software would cost more
  - Start-ups would be affected more than big companies
  - Less innovation in software industry
  - Software would be more reliable
- Making vendors responsible for harmful consequences of defects may be wrong, but...
- Consumers should not have to pay for bug fixes