Infamous Software Failures

(don't let these happen to you!)

CS A470

What is Good Software?

- Depends on your point of view
- Five perspectives
  - Transcendental view. Quality can be recognized but hard to define
  - User view. Fitness for purpose
    - Often adopted by customers or marketers
  - Manufacturing view. Conforms to specs.
  - Product view. Innards hidden, outside black-box works and is available
    - Often adopted by researchers
  - Value view. Amount someone will pay
    - Tradeoffs for risk and quality; e.g. buggy products

Product Failures

- Consequences vary
  - Loss of user confidence
  - Loss of competitiveness
  - Catastrophic event
- How many faults are there?
  - Hatton 1998, 6-30 faults per 1000 lines of code
  - DoD, 5-15 faults per 1000 lines of code
    - 1.25 hours to find, 2-9 hours to fix
  - Windows XP : 50 million lines of code
- Many dramatic faults lie in the design
  - Must understand design to predict failures

Types of Software Failure

- Process
- Real-Time anomalies
- Accuracy
- Abstractions
- Constraints
- Reuse
- Logic

Process Failure

- Human errors
  - Failures in development (e.g., poor development methodologies)
  - error in operation
- Therac-25
  - Radiation treatment machine malfunction
  - Delivers small doses of radiation through filters to treat cancers, tumors
  - Six deaths due to lethal dose of radiation before fixed

Therac-25

- Updated version of Therac-20
  - Hardware interlocks stopped machine if errors occurred
  - Therac-25 designers thought the software was good since techs never reported any problems with Therac-20
  - Software errors resulted with no ill effect, so many errors on screen they were ignored
- Therac-25 : hardware interlocks replaced with software
  - Flag: when no errors in setup, flag set to zero
  - But only 1 byte for errors, if 256 errors there was overflow back to 0
  - Machine thought tests passed when they really failed

Therac-25

- “That means that on every 256th pass through Set-Up Test, the upper collimator will not be checked and an upper collimator fault will not be detected.

  The overexposure occurred when the operator hit the "set" button at the precise moment that Class3 rolled over to zero. Thus Chkcol was not executed, and F$mal was not set to indicate the upper collimator was still in field-light position. The software turned on the full 25 MeV without the target in place and without scanning.

  …

  AECL described the technical "fix" implemented for this software flaw as simple: The program is changed so that the Class3 variable is set to some fixed nonzero value each time through Set-Up Test instead of being incremented. ”

An Investigation of the Therac-25 Accidents
Leveson & Turner

Therac-25

- Two errors here: human process and accuracy
  - Took two years to diagnose and fix
  - Lesson: Can’t separate software process from hardware
  - Need for robust software testing

Mars Climate Observer

- Observer lost 9/99
- Lockheed Martin provided thrust data in pounds, JPL entered data in Newtons
- Ground control lost contact trying to settle observer into orbit

- Process/Communications/Human error, not really a software problem

Korean Air Flight 801

- On August 6, 1997, Korean Air Flight 801 crashed into a rocky hillside in the middle of Guam. 228 of the 254 passengers on board were killed.
- Primary cause of the crash due to negligence of the flight crew, but the Minimum Safe Altitude Warning system also failed
  - FAA system responsible for warning officials when aircrafts approach too near to the ground
  - "The altitude warning system is designed to cover a circular area with a radius of 55 nautical miles (102 kilometers). However, since the software was modified, the system only covered a mile-wide circular strip that ran the circumference of that area. Flight 801 was not covered when it crashed. Black [US National Transportation Safety Board investigator] said the software was modified to stop the system from giving too many false alarms. 'The modification modified too much,' he said." (Delaney, 1997)

Real-Time Anomaly

- Example: Mars Pathfinder
  - Lander/relay for Sojourner robot
  - Onboard computer would spontaneously reset itself
  - Reported by the media as a "software glitch"
  - Used embedded real-time operating system, vxWorks

Pathfinder Problem – Priority Inversion

- Pathfinder contained an information bus
  - Data from Pathfinder's sensors, Sojourner went on bus toward earth
  - Commands from earth send along the bus to sensors
- Must schedule the bus to avoid conflicts
  - Used semaphores
  - If high-priority thread was about to block waiting for a low priority thread, there was a split-second where a medium-priority thread could jump in
  - Long-running medium priority thread kept low priority thread from running which kept the high-priority thread from running
- Good news: watchdog timer noticed thread did not finish on time, rebooted the whole system
- Noticed during testing, but assumed to be "hardware glitches". The actual data rate from Mars made the "glitch rate" much higher than in testing

Pathfinder

- Fortunate that JPL engineers left debugging code that enabled the problem to be found and remotely invoke patch
- Patch: Priority Inheritance
  - Have the low priority thread inherit the priority of the high priority thread while holding the mutex, allowing it to execute over the medium priority thread
- Such race conditions hard to find, similar problem existed with the Therac-25
- Reeves, JPL s/w engineer: “Even when you think you’ve tested everything that you can possibly imagine, you’re wrong.”

Mars Rover : Spirit

- “Spirit began acting up last week, when it stopped sending data and began rebooting its computer, resetting it roughly 130 times. At one point, the rover thought it was 2053.”

- Bug Description
  - Engineers found that the rover's 256 megabyte flash memory had retained hundreds of files containing flight data and was still juggling them along with the daily flood of new data from its activities in Mars' Gusev Crater.

Spirit

- Workaround
  - By commanding Spirit each morning into a mode that avoids using the flash memory, engineers plan to begin deleting hundreds of unneeded files to make the memory more manageable for the rover's RAM.
- WHY WASN'T THIS CAUGHT IN TEST?
  - The bug had not been detected in operational tests of the rover on Earth because the longest tests lasted only eight or nine days.

Russia: Software bug made Soyuz stray

- STAR CITY, Russia (AP) -- A computer software error likely sent a Russian spacecraft into a rare ballistic descent that subjected the three men on board to chest-crushing gravity loads that made it hard to breathe, space experts said Tuesday.

  Their capsule landed nearly 300 miles off-target in Kazakhstan. Two hours passed before anyone knew where they were or whether they were even alive.

  The capsule came in considerably steeper than planned, and the men endured more than eight times the force of gravity -- double the usual amount.

  "So it's hard to breathe and your tongue sort of slips in your head and toward the back of your throat," Bowersox said at cosmonaut training headquarters outside Moscow.
- http://www.cnn.com/2003/TECH/space/05/06/soyuz.landing.ap/index.html

Soyuz Glitch

- Slow, controlled descent requires that the guidance computer recognize what direction is “up” and where the spacecraft is in relation to its aim point far ahead. While not a particularly complex calculation, it is one that must be done with high precision and reliability.
- What happened this time was that the autopilot suddenly announced to the crew that it had forgotten where it was and which way it was headed.
- “The auto system switched to backup,” a NASA source told MSNBC.com, “which surprised them”.
- Without guidance commands, the autopilot would stabilize the spacecraft using a simple-minded backup procedure. It would send commands to steering thrusters to perform a slow roll, turning the spacecraft’s “heavy side” continuously around the dial. This had the desired effect of “nulling out” any now-unsteerable lift and let the Soyuz follow a “ballistic” descent.
- But this also meant that without the lift to stretch its flight path, the Soyuz would fall faster into thicker air. That in turn would impede the spacecraft’s forward speed even more aggressively, resulting in a deceleration about twice as high as normal and a landing far short of the planned site.

Approximation/Accuracy

- Patriot Missile Example
  - More embedded software
  - Fault in the guidance software
  - Cumulative timing fault
- Radar detects missile, calculates where the Scud will be within its range gate
- Requires accurate determination of velocity

Patriot Missile

- Patriot's internal clock: 100 ms
- Time: 24 bit integer
- Velocity: 24 bit float
- Loss of precision converting from integer to float!
  - Precision loss proportional to target's velocity and the length of time that the system is running
- When running for over 100 hours, range gate shifted by a whopping 687 meters
- Perhaps just even worse: bug known beforehand, not fixed until after incident due to lack of procedures for wartime bug fixes

Patriot Fixes

- Languages such as Ada provide fixed-point types with the convenience of floating-point with the accuracy of scaled integer arithmetic
- First validated Ada 95 compiler written for the Patriot Missile computer

Abstractions

- Y2K Bug
  - Mostly business software, some control
  - 99 to 00
  - Algorithms incorrectly interpret year 2000 as 1900
- Efficiency before correctness
  - Easy solution not necessarily the best

Y2K Problem

- A big problem because of lack of abstraction
  - Poor encapsulation of year data
  - Dates spread throughout the code without abstraction mechanisms
- Solution – better design
  - Use abstract data type for Time
  - A program can then be fixed by changing the code in only one place

Constraint Faults

- Typical examples
  - Stray pointer
  - Buffer overrun
    - Common method of overcoming security
    - Malicious code can be laid onto a string, exceed the size of an array, and place the code into memory
- Solutions
  - Recent languages such as Java, C#, Ada include constraint checking on data types
    - Provides a contract on the data type that cannot be violated
  - "Sandbox" philosophy to guard against malicious faults

Reuse

- Example: Therac-25
- Example: Ariane-5 rocket launcher
  - Rocket carried satellite as payload
  - Unexpected software exception ultimately resulted in the explosion of the rocket

Reuse w/Ariane-5

- Ariane-4
  - Successfully deployed
  - Software was optimized to avoid contract exceptions that could not possibly happen
- Ariane-5
  - Used same software as Ariane-4
  - Since it tested successfully on Ariane-4, assumed to work fine with Ariane-5, passed in simulation
  - Unexpected sensor w/overflow led to "Unhandled Exception" error

Ariane-5

- Unhandled Exception error
  - 64 bit value forced into 16 bit register
    - Not a problem with Ariane 4, but Ariane 5 was faster!
  - Switched to backup
  - Same problem
  - Ariane-5 assumed the worst and self-destructed
- Solutions
  - More testing
  - Foresight on part of developers
  - Some languages allow parameterization, generic packages

Logic Faults

- "Obvious" flaw in logic processing
- Example: AT&T Failure of 1990
  - Software upgrade of switch
    - When a switch had errors, it routed traffic to other switches while resetting itself by sending "out of service" message
    - Message caused other switches to crash, sending "out of service" message, propagating the problem
  - Traced to missing "break" statement
  - 9 hour crash, estimated $60 million lost revenue
- Solutions:
  - Testing
  - Note: C# language requires the break statement

Lessons from Faults

- Many of these faults could have been discovered through:
  - Better requirements/design specifications
    - So design your projects carefully!
  - Testing
    - Unit-level testing
    - System-wide and regression testing
      - Because a test was successful in the past doesn't mean it will stay that way!
      - Changes in one module might have subtle influence on another
      - Entire suite of tests must be re-run when a single module is changed
    - Stress testing
    - We'll have more to say on testing later…

Software Errors Cost Billions

- (Reuters) June 28, 2002 — According to a study by the U.S. Department of Commerce's National Institute of Standards and Technology (NIST), the bugs and glitches cost the U.S. economy about $59.5 billion a year.
- "The impact of software errors is enormous because virtually every business in the United States now depends on software for the development, production, distribution, and after-sales support of products and services," NIST Director Arden Bement said in a statement on Friday.
- Software users contribute about half the problem, while developers and vendors are to blame for the rest, the study said. The study also found that better testing could expose the bugs and remove bugs at the early development stage could reduce about $22.2 billion of the cost.
- "Currently, over half of all errors are not found until 'downstream' in the development process or during post-sale software use," the study said.

Who should be liable for software failures?

- In 2002 the National Academy of Sciences report from the computer and telecommunications board said that currently software makers do not have enough incentive to ensure their products are secure.

  It recommended that the US Government consider amending laws so that software makers can be held liable if their products put the public and businesses at risk.
- If software makers were held liable, the cost to consumers would likely rise dramatically
- Some action taken in European Courts
  - A Dutch judge in 9/2002 convicted Exact Holding of malpractice for selling buggy software, rejecting the argument that early versions of software are traditionally unstable.