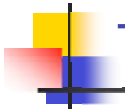
Lecture 22: Software Disasters

Kenneth M. Anderson
Software Methods and Tools
CSCI 3308 - Fall Semester, 2004



Today's Lecture

- Discuss several different software disasters to provide insights into
  - the types of errors that can occur
  - the costs associated with them
- Examples
  - Mars Climate Orbiter
  - Mars Polar Lander
  - Patriot Missile Defense System
  - Ariane 5

Mars Climate Orbiter

- Science Objectives
  - Monitor climate changes
  - Serve as relay for Mars Polar Lander
- Costs for Climate Orbiter and Polar Lander combined
  - Spacecraft Development - 193.1 million
  - Launch - 91.7 million
  - Mission and Operations - 42.8 million
  - Total - 327.6 million

Mars Climate Orbiter, continued

- Supposed to enter Martian atmosphere "at a high trajectory" and "lightly aerobrake" to achieve orbit, which uses less fuel
- Due to a conversion error in which commands to the spacecraft were sent in English units rather than metric units, the spacecraft entered the atmosphere at "a trajectory 170km lower than planned"
  - The spacecraft hit the atmosphere earlier than was planned and was thus traveling too fast; this led to the destruction of the spacecraft
  - Since the Polar Lander was also lost, the combined cost of the project stands at 327.6 million dollars
  - Compare to Mars Observer (lost in 1989): 4 billion dollars!
- Unofficially, the problem had been detected but due to politics between the development team and JPL, a fix was never deployed

Mars Climate Orbiter, continued

- The official report cited the following “contributing factors” to the loss of the spacecraft
  - undetected errors in ground-based models of the spacecraft
  - the operational navigational team was not fully informed on the details of the way that Mars Climate Orbiter was pointed in space
  - a final, optional engine firing to raise the spacecraft’s path relative to Mars before its arrival was considered but not performed

Mars Climate Orbiter, continued

- Contributing Factors, continued
  - the systems engineering function within the project that is supposed to track and double-check all interconnected aspects of the mission was not robust enough
    - this was exacerbated by the first-time handover of a Mars-bound spacecraft by the team that constructed and launched the vehicle to a new multi-mission operations team (this is the “politics” part!)
  - some communications channels among project engineering groups were too informal (e.g. not documented!)

Mars Climate Orbiter, continued

- Contributing Factors, continued
  - the small mission navigation team was oversubscribed and its work did not receive peer review by independent experts
  - personnel were not trained sufficiently in areas such as the relationship between the operation of the mission and its detailed navigational characteristics, or the process of filing formal anomaly reports
  - the process to verify / validate certain engineering requirements and the technical interfaces between some project groups, and between the project and its prime mission contractor was inadequate

Mars Climate Orbiter, summary

- One Technical Problem
  - failed conversion of units
- Many Process and Social Problems
  - No review (e.g. verification), insufficient training, informal processes in place, formal processes ignored
- Led to a destroyed spacecraft

Mars Polar Lander

- Part of the same project as the Mars Climate Observer
- Last communication with spacecraft occurred just prior to its entry into the Martian atmosphere
- Loss of spacecraft can be attributed to the lack of integration testing
  - e.g. the failure was not detected by “module” testing
  - an integrated test across subsystems was required to detect the problem

Details

- Module Test 1
  - Give command to deploy spacecraft’s legs
    - Legs deploy
      - Test Passed!
- Module Test 2
  - When spacecraft detects “jolt” of landing on the surface of Mars, turn engine off
    - Simulated “jolt” detected, engine shuts off
      - Test Passed!

Details, continued

- What actually happened
  - Spacecraft enters atmosphere
  - Legs deploy and “jolt” the craft
  - “Jolt” detected and engine shuts off
- The problem
  - Spacecraft was still miles above the surface of Mars!
  - Spacecraft crashes into Mars and is destroyed

Mars Polar Lander, summary

- Clear demonstration of the importance of integration testing
  - If the team testing the deployment of the legs had conducted the test while also testing the flight software, the “bug” may have been detected
  - Unfortunately, with the “faster, better, cheaper” philosophy of NASA at the time, integration testing was deemed too expensive and was not conducted in a comprehensive fashion

Credit Where Credit is Due

- Information for the rest of the lecture comes from
  - <http://dutita0.twi.tudelft.nl/users/vuik/wi211/disasters.html>
- and
  - <http://www.eiffel.com/doc/manuals/technology/contract/ariane/>
- Additional Info on Ariane 5
  - <http://www.math.ufl.edu/~cws/3114/ariane-siam.html>

Patriot Missile Defense System

- On February 25, 1991, during the Gulf War, a patriot missile failed to intercept an incoming Iraqi Scud missile
  - The cost of this failure was the lives of 28 soldiers when the Scud missile struck a military barracks
- The cause of the failure was a numerical error in the Patriot's operating system to correctly calculate "time since boot"
- This caused the system's time to get out of synch with actual time and led to a failure in the system designed to track the Scud missile's position in the air

Details

- The system's internal clock measured time in tenths of seconds
  - Actual time was reported in seconds by multiplying the internal clock's value by 1/10
  - This calculation was performed using a 24-bit fixed point register

Details, continued

- The problem?
  - 1/10 has a non-terminating binary expansion, so its actual value was chopped to fit into a 24-bit register
  - This introduces an error equal to 0.000000095
  - After running for 100 hours, this error means that the system is 0.34 seconds out of sync with reality
- What's wrong with that?
  - Scud Missiles travel at 1676 meters per second!
    - In 0.34 seconds, they travel half a kilometer (~0.3 of a mile)!
  - Without an accurate location, the Patriot missile had no chance to intercept the missile

Patriot Missile System, summary

- Software Engineering Issues
  - Numerical algorithm at an extremely low level led to total system failure and loss of human life
  - But there was also a maintenance related problem
    - The error was known, and a fix had been applied in some parts of the software but not others!
  - Inadequate maintenance process!
    - A complete set of regression tests, if applied after every modification, may have detected the problem

Ariane 5 Disaster

- On June 4, 1996, after 7 billion dollars of development, an unmanned Ariane 5 rocket exploded just forty seconds after lift-off
  - The rocket and its cargo were valued at $500 million for a total cost of 7.5 billion dollars!
- The error was traced to a software component in the Inertial Reference System that had been reused from the Ariane 4 flight software
  - The reused component was more than 10 years old and had flown successfully on numerous Ariane 4 flights
  - The problem => certain assumptions changed between the Ariane 4 and the Ariane 5 and the software was not updated in response

Ariane 5, background info

- The flight software was written in Ada which has a first class exception construct
  - (it predates C++ and Java in this regard)
- If an exception is thrown but not caught, the error will “percolate” up through the call stack and will eventually terminate the entire system

Ariane 5, the details

- The failure of the Ariane 5 can be traced to the conversion of a 64-bit integer to a 16-bit signed integer
  - The 64-bit value was greater than 2^{15} which caused an exception to be generated
  - This exception was not caught and it caused the termination of the flight control software 37 seconds into the launch
  - The rocket shortly thereafter (3 seconds) lost control and was destroyed

More information

- Jean-Marc Jézéquel and Bertrand Meyer wrote a paper that traces the problem to an inappropriate reuse of a 10-year old software component
  - They reveal that one “vexing” aspect of this disaster is that the error occurred in a software system that was not needed during launch!
    - The calculation was supposed to be stopped 9 seconds before launch, but the inertial reference system had been reset during a hold in the countdown and its initialization sequence proceeded during launch.
      - This is what caused the rocket to veer off course… the initialization sequence was sending random sequences of 1s and 0s to the flight control software, which was interpreting them as commands to fire various sets of booster jets in completely random patterns!

Details, continued

- Their paper reveals that sufficient software dev. processes were in place and the system that caused the error had even been reviewed extensively before launch
  - exception handlers had been placed around 4 of 7 variables; unfortunately, the data conversion error occurred in one of the 3 unprotected variables
  - why leave 3 variables unprotected? Performance! If you add exception handling code, you slow the performance of the system
  - plus, the developers had an analysis that showed that overflow could not occur with the 3 unprotected variables
  - so they had good reason to leave them unprotected

Details, continued

- The problem?
  - The overflow analysis was conducted for the Ariane 4, not the Ariane 5
    - Its prediction that overflow could not occur for the three unprotected variables was no longer valid!
- So, it was a reuse error!

Ariane 5, summary

- The authors conclude that “Design by Contract” was needed in this situation
  - In particular, the component needed to specify a “contract” with its users; one aspect of this contract is specifying the legal input values
  - If the component had done something similar to an assert construct like this
    - proc foo(actual_value: int)
      - assert(actual_value <= maximum_value)
  - The authors argue that the error may well have been detected during system test; they further argue that such “contracts” should be a first-class, required programming language construct; not an optional construct that few use