CRICOS PROVIDER 00123M

Faculty of ECMS / School of Computer Science

Software Engineering & Project
Safety Critical Systems Case Studies

Safety Critical Systems Case Studies

Lecture 14

Safety Critical SE

- As software is used more widely, safety has become an important issue.
- Safety aspects of software are poorly understood at the moment.
- Software itself is poorly understood still.
- Safety analysis cannot be usually undertaken gradually – a whole system approach is normally required.
- Safety analysis is context sensitive:
  - a system "safe" in one context may not be in another
  - E.g. a computer controlled ventilation system for use in glasshouses might not be suitable for use in a multi-story office block…

Case Study 1 – F16A

“The world’s most sophisticated, foolproof and deadly combat aircraft yet.”

- Computer monitoring of all critical operations.
  - computer would ignore pilot directives if they would result in operation outside pre-set envelope.
- Failure 1: gear-up while landed
  - solution: software now checks “squat switches”
- Failure 2: gear-down while cruising (gear down at > 300Knots is fatal)
  - solution: software now checks airspeed
- Failure 3: gear won’t come up
  - solution: correct race condition introduced by failure 2 fix which latched boolean flags for squat switches.

Morals:

- A gradual approach is clearly ineffective,
- “Quick Fixes” often are not...

Mid-air collision – August 2013

Two F-16C Falcon single seat aircraft, assigned to the 121 FS, D.C. Air National Guard, were involved in a mid-air collision August 1, 2013.

"Training is the foundation for everything we do," said Brig. Gen. Sasseville.
"The mishap is under investigation, and our findings will be used to continually improve our operations to provide the safest possible training."

Human error or software error?
What could have been done differently?
Suggestions…?

Case Study 2 – London Ambulance DS

- In the late 1980s work began on an Automated Ambulance Dispatch System
- Upon receiving calls (still logged by operators), Ambulances would be immediately dispatched, based on location and availability
- System included:
  - Automatic vehicle locating system (AVLS)
  - Support for automatic communication with ambulances
- System went live in 1992
  - Incomplete
  - Largely untested
  - Political pressures to hasten delivery

London Ambulance DS

- Failure of the system
  - AVLS was unable to keep track of location and availability of ambulances
  - Units dispatched non-optimally
  - Multiple units dispatched to some locations
  - Lead to many exception messages
    - Slowed system down – making the situation even worse
    - Exception messages scrolled of screen, so operators unable to take appropriate action
  - Repeated calls lead to further congestion
  - The system was put into a semi-manual mode
    - Problems persisted, until after 8 days the system completely froze
    - Return to fully manual operations, using tape recorded messages
- Consequences
  - As many as 20 deaths were thought to have been attributed (in some part) to ambulances arriving late or not at all
  - CEO of LAS resigned

London Ambulance DS

- What went wrong?
  - The software was incomplete and mostly untested
    - Rushed into service by political pressures
    - Should have first been tested on a smaller scale
  - Unjustified assumptions made during specification
  - Lack of consultation with end-users during development
  - Lack of quality
    - Poor user interface
    - Lack of robustness
    - Poor performance

Case Study 3 – Patriot Missile

- Overview:
  - One missile battery failed to lock onto a scud missile which subsequently killed personnel at a Saudi military camp in Dhahran
- Cause:
  - Time interval calculations were imprecise, due to the 24 bit registers in the hardware.
  - Why did the programmers miss this?
  - It doesn't matter till the system has been up continuously for many hours, 30 minutes of "testing" didn't detect this.
- Moral:
  - You need to consider the hardware limitations too.
  - You need to be aware of the implicit assumptions you are making

Case Study 4 – Therac 25

- A radio-therapy machine which has killed several patients by massive overdose.
- The Therac–25 is a classic example of a poorly engineered software system.
- Implemented on a PDP-11 system, written in assembly language.
- Many of the problems result from race conditions due to the use of shared structures with no synchronisation.

Moral: lack of whole system testing and methodical testing overall.

Case Study 5 - A320-200 Airbus: Warsaw 1993

- Lufthansa flight 2904, September 14 1993
- Crashed into barriers at the end of runway
- Death of 1 passenger and 1 crew member

Causes

- Weather conditions
  - Strong cross wind and lots of water on runway
- The landing
  - Pilots approached runway at greater than normal speed
  - Aircraft landed long, with slight tilt towards one side of landing gear
- Braking logic (software)
  - Prevented application of brakes until both sides of landing gear were exerting sufficient force
  - Pilots had no way of overriding braking logic

Lessons learnt

- Poor design of braking logic was the contributing factor
  - Can probably be traced back to requirements
  - When designing a system for safety must consider ALL conditions that it will be used in
- Need to question the practice of not allowing user overriding of software control
  - Not a clear cut issue though – pilots may have made the situation worse??
  - This is still the policy for Airbus, differs from Boeing's policy

Case Study 6 - Ariane 5

June 4th 1996 - Total failure of the Ariane 5 launcher on its maiden flight

- A European rocket designed to launch commercial payloads (e.g.communications satellites, etc.) into Earth orbit
- Successor to the successful Ariane 4 launchers
- Ariane 5 can carry a heavier payload than Ariane 4
- More details are available at
  - https://around.com/ariane.html

esa
ariane
esa
ariane
esa
© 2003 - ESA - CNES - ARIANESPACE / photo service Optique Vidéo CSG

Launcher failure

- Incorrect control signals were sent to the engines and these swivelled so that unsustainable stresses were imposed on the rocket
- It started to break up and self-destructed

Bang! $600 million US

The problem

- The attitude and trajectory of the rocket are measured by a computer-based inertial reference system. This transmits commands to the engines to maintain altitude and direction
- The software failed and this system and the backup system shut down
- Diagnostic commands were transmitted to the engines which interpreted them as real data and which swivelled to an extreme position

Software failure

- Software failure occurred when an attempt to convert a 64-bit floating point number to a signed 16-bit integer caused the number to overflow.
- There was no exception handler associated with the conversion so the system exception management facilities were invoked.
- These shut down the software.
- The backup software was a copy, it behaved in exactly the same way…!

The software was originally written for the Ariane 4 where efficiency considerations (the computer running the software had an 80% maximum workload requirement) led to four variables being protected with a handler while three others, including the horizontal bias variable, were left unprotected because it was thought that they were "physically limited or that there was a large margin of error"

Avoidable?

- The software that failed was reused from the Ariane 4 launch vehicle.
- The computation that resulted in overflow was not used by Ariane 5.
- Decisions were made
  - Not to remove the facility as this could introduce new faults
  - Not to test for overflow exceptions because the processor was heavily loaded. For dependability reasons, it was thought desirable to have some spare processor capacity

Why not Ariane 4?

- The physical characteristics of Ariane 4 (A smaller vehicle) are such that it has a lower initial acceleration and build up of horizontal velocity than Ariane 5

- The value of the variable on Ariane 4 could never reach a level that caused overflow during the launch period.

Validation failure?

- As the facility that failed was not required for Ariane 5, there was no requirement associated with it.
- As there was no associated requirement, there were no tests of that part of the software and hence no possibility of discovering the problem.
- During system testing, simulators of the inertial reference system computers were used. These did not generate the error as there was no requirement!

Review failure

- The design and code of all software should be reviewed for problems during the development process
- Either:
  - The inertial reference system software was not reviewed because it had been used in a previous version
  - The review failed to expose the problem
    - or that the test coverage would not reveal the problem
  - The review failed to appreciate the consequences of system shutdown during a launch

Failure Review Conclusions

- The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift- off). This loss of information was due to specification and design errors in the software of the inertial reference system.
- The extensive reviews and tests carried out during the Ariane 5 Development Programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure.

http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html

Lessons learned

- Don’t run software in critical systems unless it is actually needed
- As well as testing for what the system should do, you may also have to test for what the system should not do
- Do not have a default exception handling response which is system shut-down in systems that have no fail-safe state
- In critical computations, always return best effort values even if the absolutely correct values cannot be computed
- Wherever possible, use real equipment and not simulations
- Improve the review process to include external participants and review all assumptions made in the code

Avoidable failure

- The designer’s of Ariane 5 made a critical and elementary error.
- They designed a system where a single component failure could cause the entire system to fail

Food for thought

- http://www.ibm.com/developerworks/rational/library/agile-analysis-practices-safety-critical-development/
- http://www.ibm.com/developerworks/rational/library/safety-related-software-development/index.html