Lecture 2
Why Does Software Fail?

- Background
  - What is Software Engineering?
  - What causes system failures?
  - What is the role of good engineering practice?
- Software failures vs. hardware failures
  - Example: Ariane-5 Flight 501
  - Example: Mars Exploration Rover (MER) Mission
- Lessons learned

Defining Software Engineering

- "Engineering…
  - … creates cost-effective solutions to practical problems by applying scientific knowledge to build ***things*** in the service of humankind"
- Software Engineering…
  - … when the ***things*** contain software
- BUT:
  - Pure software is useless!
    - **… software exists only as a part of a system**
  - Software is invisible, intangible, abstract
  - There are no physical laws underlying software behavior
  - There are no physical constraints on software complexity
  - Software never wears out
    - **… traditional reliability measures do not apply**
  - Software can be replicated perfectly
    - **… no manufacturing variability**

Failures and catastrophes

- (Hardware) system components often fail
  - Parts wear out
  - Wires and joints come loose
  - Components get used for things they were not designed for
  - Designs don't work the way they should
- Point failures typically do not lead to catastrophe
  - There are…
    - **backup systems**
    - **fault-tolerant designs**
    - **redundancy**
    - **certification using safety factors**
- Good engineering practice prevents accidents
  - Failure analysis
  - Reliability estimation
  - Checks and balances
- *But how does this work in Software Engineering?*

Ariane-5 Flight 501

- Background
  - The reusable launch vehicle of the European Space Agency (ESA)
  - Ariane-4 a major success
  - Ariane-5 developed for larger payloads
- Launched
  - 4 June 1996
- Mission
  - $500 million payload to be delivered to orbit
- Fate
  - Veered off course during launch
  - Self-destructed approx. 40 sec after launch
- Cause
  - Unhandled floating-point exception in Ada code

Ariane-5
Events

- Locus of error
  - Platform alignment software (part of the Inertial Reference System, SRI)
  - This software only produces meaningful results prior to launch
  - Still operational 40 sec after launch
- Cause of error
  - Ada exception code raised and not handled
    - Converting 64-bit floating point to 16-bit signed integer for Horizontal Bias (BH)
    - Conversion worked fine in Ariane-4; overflow occurred in Ariane-5
  - Requirements state that computer shut down if unhandled exception occurs
- Launch+30s: SRIs fail
  - Backup SRI shuts down first
  - Active SRI shuts down 50ms later for same reason
- Launch+31s: On-Board Computer (OBC) receives data from active SRI
  - Diagnostic bit pattern interpreted as flight data (nonsensical input data)
  - OBC commands full nozzle deflections
  - Rocket veers off course
- Launch+33s: Launcher starts to disintegrate
  - Self-destruct triggers

Ariane-5
Why did this failure occur?

- Why was Platform Alignment still active after launch?
  - SRI software reused from Ariane-4
  - Ariane-5 was faster and used different timing parameters
- Why was there no exception handler?
  - To reduce the processor workload to below 80%
- Why wasn't the design modified for Ariane-5?
  - Not considered wise to change software that worked well for Ariane-4
- Why did the SRIs shut down?
  - They assumed faults are random hardware failures, hence should switch to backup
- Why was the error not caught in unit testing?
  - No trajectory data for Ariane-5 was provided in the requirements for SRIs
- Why was the error not caught in integration testing?
  - Full integration testing considered too expensive
  - SRIs were considered to be fully certified
  - Integration testing used simulations of the SRIs
- Why was the error not caught by inspection?
  - The implementation assumptions were not documented
- Why did the OBC diagnostic data as flight data?
  - They assumed this couldn't happen(!!)

Ariane-5
Summary

- Factors that lead to the Ariane-5 accident:
  - Didn't test all the specification assumptions
  - Insufficient test data
  - Lack of integration testing
  - Lack of expertise at inspections
  - Poor communication between teams
  - Commercial/management pressures took priority
  - Reused old code without checking newer assumptions
  - "Redundant" design not redundant

Mars Exploration Rover (MER)

- Background
  - The latest Mars mission of NASA
  - In spite of many problems, it's a major success
- Timeline
  - MER-A "Spirit"
    - Launched on Jun 10, 2003
    - Landed in Gusev Crater on Jan 4, 2004
  - MER-B "Opportunity"
    - Launched on Jul 7, 2003
    - Landed on Meridiani Planum on Jan 24, 2004
  - Designed to work for at least 90 days
  - Mars disappeared behind the Sun and out of the audio range on Sep 13, 2004
  - The rovers were still alive and running on Sep 13 (!)

MER
Events

- Jan 21, 2004: Spirit stops communicating with the Earth
  - It reboots itself continuously
- Jan 26, 2004: The team discovers the cause
  - Corruption inside Spirit's on-board flash memory
  - The rovers are put into "deep sleep" until a patch is deployed
- February, 2004: The team beams up a software patch
  - Both rovers are up and running
- April, 2004: The team beams up another, non-critical software patch
- Locus of error
  - The DOS-like file allocation system, stored in the rover's flash memory
- Cause of error
  - Erasing files in DOS does not remove the file entry in directory, but marks it as "erased" and makes it available for further use
  - Creating many files at a time causes the directory to grow, without ever getting smaller (unless it's erased)
  - The flash memory was mirrored in the system's RAM, leading to overflows later on

MER
Why did this failure occur?

- Why was a DOS-like file system used?
  - It's simple, fast, and it was successfully used on prior missions
  - Extreme radiation conditions on Mars require a slow CPU clock
    - Software cannot be too complex
- Why was the file system overloaded?
  - A large number of tiny files were created during data acquisition
- Why did the operating system crash?
  - The data stored in flash memory (including file system) was mirrored in the system's RAM
  - The RAM was nearly full, and further dynamic memory allocations resulted in a crash
- Why was the error not caught in prior testing?
  - NASA's tests only allowed for the addition of a small number of data files
- How was the error fixed?
  - The dynamic allocation feature was disabled
  - The flash memory was erased
  - A file system monitoring utility was installed

MER
Summary

- Factors that lead to the flash memory problems:
  - Didn't test all specification assumptions
  - Insufficient test data
  - Lack of integration testing
  - Poor communication between teams
  - Reused code without checking assumptions
  - Others
    - … to be disclosed by NASA (?)

Lessons learned

- Failures can usually be traced to a single root cause
  - … but good engineering practice should prevent system failures
  - The real problems are failures of:
    - **testing and inspection process**
    - **problem reporting and tracking**
    - **lack of expertise**
    - **inadequate resources**
    - **etc.**
  - In most cases, it takes a failure of both engineering practice and management to cause a system failure
- Reliable software depends not on writing flawless code, but on how good are we at:
  - Communication (sharing information between teams)
  - Management (of resources and skills)
  - Verification and validation
  - Risk identification and tracking
  - Questioning assumptions

Readings

- Hans van Vliet's book, Chapter 1
  - Read all of it
  - Pay attention to the code of ethics
- Ariane-5
  - Information about ESA's launches
    - http://www.esa.int/export/esaLA/launchers.html
  - Flight 501 inquiry report & Press release:
    - http://www.esrin.esa.it/htdocs/tidc/Press/Press96/press33.html
- Mars Exploration Rover (MER) Mission
  - Information about NASA's MER Mission
    - http://marsrovers.jpl.nasa.gov/home/
  - DOS glitch nearly killed Mars Rover
    - http://www.extremetech.com/article2/0,1558,1638764,00.asp