EDA122/DIT061 Fault-Tolerant Computer Systems

Welcome to Lecture 4

Development faults

Outline

- Case study: Ariane 501 Disaster
- Software redundancy
- Design diversity
  - N-version programming
  - Recovery blocks

Ariane 5 Disaster

”On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in failure. Only about 40 seconds after the initiation of the flight sequence, at an altitude of about 3700 m, the launcher veered off its flight path, broke up and exploded.”

(From J.L. Lions, et al, ARIANE 5 Flight 501 failure, http://www.esrin.esa.it/tidc/Press/Press96/ariane5rep.html)

Ariane 5 Disaster

- Development cost ~8 billion euros
- Loss of ~500 million euros
- One year delay of the Ariane program

Ariane 5 Flight Control System

- An Inertial Reference System (SRI) measures the attitude of the launcher and its movements in space. The SRI calculates angles and velocities which are sent to the On-Board Computer (OBC).
- The OBC executes the flight control program and controls the nozzles of the solid boosters and the Vulcain Cryogenic engine.
- There are two SRIs with identical hardware and software operating, one active and one in hot stand-by.

Engine Nozzles

Ariane 5 Disaster

From the press release issued by ESA on July 23, 1996: (http://www.esrin.esa.it/tidc/Press/Press96/press33.html)

The Inquiry Board report begins by presenting the causes of the failure, analysis of the flight data having indicated:

- nominal behaviour of the launcher up to H0 + 36 seconds;
- failure of the back-up Inertial Reference System followed immediately by failure of the active Inertial Reference System;
- swivelling into the extreme position of the nozzles of the two solid boosters and, slightly later, of the Vulcain engine, causing the launcher to veer abruptly;
- self-destruction of the launcher correctly triggered by the rupture of the electrical links between the solid boosters and the core stage.

Ariane 5 Disaster
Chain of events, tracing backwards in time

(From J.L. Lions, et al, ARIANE 5 Flight 501 failure, http://www.esrin.esa.it/tidc/Press/Press96/ariane5rep.html)

- The launcher started to disintegrate at about H0 + 39 seconds because of high aerodynamic loads due to an angle of attack of more than 20 degrees that led to separation of the boosters from the main stage, in turn triggering the self-destruct system of the launcher.
- This angle of attack was caused by full nozzle deflections of the solid boosters and the Vulcain main engine.

Ariane 5 Disaster
Chain of events, tracing backwards in time

- These nozzle deflections were commanded by the On-Board Computer (OBC) software on the basis of data transmitted by the Inertial Reference System 2 (SRI 2). Part of these data at that time did not contain proper flight data, but showed a diagnostic bit pattern of the computer of the SRI 2, which was interpreted as flight data.
- The reason why the active SRI 2 did not send correct attitude data was that the unit had declared a failure due to a software exception.
- The OBC could not switch to the back-up SRI 1 because that unit had already ceased to function during the previous data cycle (72 milliseconds period) for the same reason as SRI 2

Ariane 5 Disaster
Chain of events, tracing backwards in time

- The internal SRI software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error. The data conversion instructions (in Ada code) were not protected from causing an Operand Error, although other conversions of comparable variables in the same place in the code were protected.
- The error occurred in a part of the software that only performs alignment of the strap-down inertial platform. This software module computes meaningful results only before lift-off. As soon as the launcher lifts off, this function serves no purpose.

Ariane 5 Disaster
Chain of events, tracing backwards in time

- The alignment function is operative for 50 seconds after starting of the Flight Mode of the SRIs which occurs at H0 - 3 seconds for Ariane 5. Consequently, when lift-off occurs, the function continues for approx. 40 seconds of flight. This time sequence is based on a requirement of Ariane 4 and is not required for Ariane 5.
- The Operand Error occurred due to an unexpected high value of an internal alignment function result called BH, Horizontal Bias, related to the horizontal velocity sensed by the platform. This value is calculated as an indicator for alignment precision over time.
- The value of BH was much higher than expected because the early part of the trajectory of Ariane 5 differs from that of Ariane 4 and results in considerably higher horizontal velocity values.

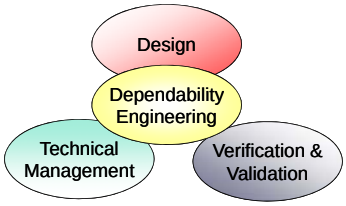
Lessons Learned from the Ariane 5 Disaster

- Both random faults and systematic (development/design) faults must be considered.
- Do not expect software, which has proven to be reliable in one environment, to be reliable in another environment.
- Ensure that system tests that are realistic.
- Use an "intelligent" error handling strategy
  - Shut down non-critical tasks (processes) that fail.
  - Try to recover from errors that occur in critical tasks before shutting down the unit.
  - Enforce omission failures instead of crash failures, whenever possible.

Example of an "intelligent" error handling strategy

- Provide a mechanism to separate *critical services* and *non-critical services*
- Employ a "*never give up*" strategy for critical services
  - Provide error recovery for critical services
  - Make the system resilient to *omission failures* (*temporary service failures*)
  - Provide support for graceful degradation, if possible.
  - Enforce *crash failures* only as a last resort
- Shut down non-critical services that fail.

Competence Areas



Discussion

Like many disasters, the Ariane 501 failure was caused by a *lack of knowledge and insight*

In which of the competence areas

- ***Design***
- ***Verification and Validation***
- ***Technical Management***

did the Ariane 5 project lack knowledge and insight?

Recommendations made by the Inquiry Board (1-4)

- **R1** Switch off the alignment function of the inertial reference system immediately after lift-off. More generally, no software function should run during flight unless it is needed.
- **R2** Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform complete, closed-loop, system testing. Complete simulations must take place before any mission. A high test coverage has to be obtained.
- **R3** Do not allow any sensor, such as the inertial reference system, to stop sending best effort data.
- **R4** Organize, for each item of equipment incorporating software, a specific software qualification review. The Industrial Architect shall take part in these reviews and report on complete system testing performed with the equipment. All restrictions on use of the equipment shall be made explicit for the Review Board. Make all critical software a Configuration Controlled Item (CCI).

Recommendations made by the Inquiry Board (5-7)

- **R5** Review all flight software (including embedded software), and in particular :
  - Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment.
  - Verify the range of values taken by any internal or communication variables in the software.
  - Solutions to potential problems in the on-board computer software, paying particular attention to on-board computer switch over, shall be proposed by the project team and reviewed by a group of external experts, who shall report to the on-board computer Qualification Board.
- **R6** Wherever technically feasible, consider confining exceptions to tasks and devise backup capabilities.
- **R7** Provide more data to the telemetry upon failure of any component, so that recovering equipment will be less essential.

Recommendations made by the Inquiry Board (8-12)

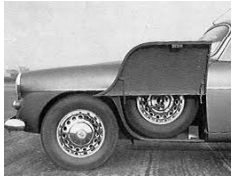
- **R8** Reconsider the definition of critical components, taking failures of software origin into account (particularly single point failures).
- **R9** Include external (to the project) participants when reviewing specifications, code and justification documents. Make sure that these reviews consider the substance of arguments, rather than check that verifications have been made.
- **R10** Include trajectory data in specifications and test requirements.
- **R11** Review the test coverage of existing equipment and extend it where it is deemed necessary.
- **R12** Give the justification documents the same attention as code. Improve the technique for keeping code and its justifications consistent.

Recommendations made by the Inquiry Board (13-14)

- **R13** Set up a team that will prepare the procedure for qualifying software, propose stringent rules for confirming such qualification, and ascertain that specification, verification and testing of software are of a consistently high quality in the Ariane 5 programme. Including external RAMS experts is to be considered.
- **R14** A more transparent organisation of the cooperation among the partners in the Ariane 5 programme must be considered. Close engineering cooperation, with clear cut authority and responsibility, is needed to achieve system coherence, with simple and clear interfaces between partners.

Fault-Tolerance – How?

- By introducing ***redundancy*** (extra resources)
- Forms of redundancy
  - hardware redundancy
  - software redundancy
  - time redundancy
  - information redundancy



Software redundancy

- Software redundancy techniques can be divided in two major classes:
  - With diversity
    - Aim is to tolerate software development faults
    - Design diversity
    - Data diversity
  - Without diversity
    - Aim is to handle errors of any origin (physical faults, development faults, operator faults)

What is Software Fault Tolerance?

The term ”software fault tolerance” can mean two things:

1. ”the tolerance of software development faults”, or
2. ”the tolerance of faults by the use of software”

Definition 1 is more commonly used.
Definition 2 is used by N. Storey (author of the course book).

The term ”software redundancy” corresponds to definition 2.

Design Diversity

Design diversity is used to tolerate development faults in hardware and software

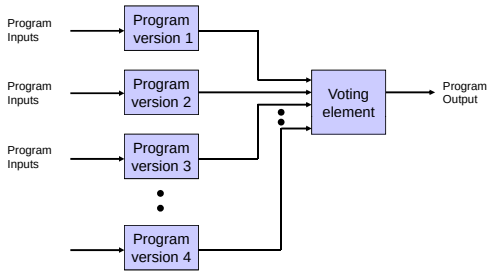
Two techniques for tolerating software design faults:

- N-version programming
- Recovery blocks

N-version programming

- Uses majority voting on results produced by N program versions
- Program versions are developed by different teams of programmers
- Assumes that programs fail independently
- Resembles hardware voting redundancy

N-version programming



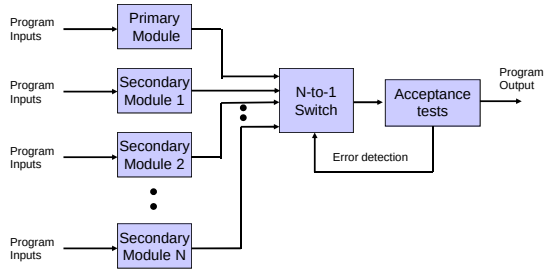
Ensuring independence in N-version programming

- Use different design teams for each version
- Use diverse specifications
- Prevent cooperation among design teams
- Use diverse programming languages, compilers, development tools, etc.
- …

Recovery Blocks

- Uses one primary software module and one or several secondary (back-up) software modules
- Assumes that program failures can be detected by acceptance tests
- Executes only the primary module under error-free conditions
- Resembles dynamic hardware redundancy

Recovery blocks



Construction of acceptance tests

- An acceptance test is a software implemented check designed to detect errors in the results produced by a primary or a secondary module
- Acceptance tests often relies on application specific information
- An acceptance test is similar to a software assertion (a.k.a. executable assertion).

Software assertions

- Executes consistency checks on application data or operating system data
- Implemented in software
- Automatic generation
  - E.g., run-time type checking generated by compiler
- Manual generation
  - E.g., application programmer inserts checks on a valid temperature range, acceleration, etc.

Examples of how acceptance tests/ software assertions can be constructed

- Satisfaction of requirements
  - Inversion of mathematical functions; e.g. squaring the result of a square-root operation to see if it equals the original operand
  - Checking sort functions; result should have elements in descending order
  - …
- Reasonable checks
  - Checking physical constraints; e.g. speed, pressure, etc
  - Checking sequence of application states
  - …

Comparison of N-version programming and Recovery blocks

N-version programming
- Applied at the program level
- Runs N programs at the same time
- Resembles voting redundancy in hardware (static redundancy)
- Assumes that independence among program versions is achieved by random differences in programming style among programmers

Recovery blocks
- Applied at the module (subprogram) level
- Runs only the primary module under error-free conditions
- Resembles standby redundancy in hardware (dynamic redundancy)
- Independence is achieved by deliberately designing the primary and secondary modules to be as different as possible

Summary

- Ariane 5 case study:
  - Don't forget to consider development faults!
  - Do not expect software which has proven reliable in one environment to be reliable in another environment.
- N-version programming
  - Relies on voting to mask software failures
  - Uses "accidental" redundancy
- Recovery blocks
  - Relies on acceptance tests and reconfiguration
  - Uses "deliberate" redundancy

Overview of Lecture 5

- Markov chain models
  - Hot standby system
  - Cold standby system
  - Coverage factor
  - Dormancy factor
- Read *before* the lecture:
  - Storey: Section 7.2 Markov models (pages 183 - 186)
  - Lecture slides