Verification & Validation of Safety Critical Software

Dr Peter Lindsay
Assistant Director
Software Verification Research Centre
School of Information Technology
The University of Queensland
The increasing trend towards systems integration, and increased automation of critical functions which were once performed by humans, means that *more and more reliance is placed on software.*

Procurers of safety-critical systems are becoming more aware of the need for appropriate levels of safety assurance, and are increasingly requiring system developers to produce a **Safety Case** to document the reasons why a system is safe to be operated.
This talk looks at recent and emerging standards for safety-critical software, and will introduce listeners to the key principles of safety assurance, including:

- hazard and risk analysis
- safety integrity levels
- the structure and content of safety cases
- management of the safety process
Computer Aided Disasters

- **Therac 25** (1985-87, N. America) radiation therapy machine delivers severe radiation overdoses (x6)
- **London Ambulance Service** (1992) 20+ die unnecessarily when dispatch system fails
- **USS Vincennes** (1988) shoots down Iran Air airliner after faulty identification
- **Airbus A320** (1988-) various crashes
- **Ariane 5** (1996) software exception causes self-destruct
- etc

See [http://www.comlab.ox.ac.uk/archive/safety.html](http://www.comlab.ox.ac.uk/archive/safety.html)
What’s Different About Software?

Broadly speaking, traditional safety engineering is concerned with physical failures:

- e.g. wear-out, corrosion, faulty manufacture
- mitigations include: well-tried designs, safety margins, redundant components, inspection, maintenance
- this has little relevance for software

On the other hand, software is typically:

- novel, complex, highly input-sensitive, not designed by domain experts

Software demands a new approach to safety engineering
Define main terms & concepts in safety engineering as they relate to software:
  – hazards, risk, safety integrity levels, etc

Explain the basic principles of safety management & the safety lifecycle for software systems

Outline 3 important safety analysis techniques
  – Failure Modes Effects Analysis (FMEA)
  – Fault Tree Analysis (FTA)
  – Hazard and Operability Studies (HAZOP)

Summary
Reference Material

- **Def(Aust) 5679** Australian Defence Standard for Procurement of Computer-based Safety-critical Systems
- **UK MOD 00-55, 00-56, 00-58** Standards for software development and hazard analysis of safety-critical systems
- **Nancy Leveson** *Safeware: System Safety and Computers*
A system is **unsafe** if it can cause unacceptable harm.

**Harm**: loss of life, injury, damage to the environment, etc.

Safety is a *whole system* issue
- only physical objects can cause harm
- need to consider all system components: software, hardware, operators, procedures, infrastructure, …

Safety is a *whole lifecycle* issue
- from concept through to decommissioning

Safety and reliability are two different things
**Hazard**: a situation with the potential for harm

- Hazards are a *state of the system*
  - scope of system needs careful definition
  - other factors (outside system control) may affect whether hazard leads to an accident

**Failure mode**: the way in which something fails
Risk

- Absolute safety is generally unachievable
  - instead, aim for *acceptable risk*

- **Risk**: a combination of the *severity* of consequences & *likelihood* of occurrence

- **Severity**: the possible extent of harm

- **Likelihood**: the probability/frequency of occurrence
  - eg. probability of $10^{-6}$ that X fails on request;
    mean-time-to-failure is 2 years;
    probability of failure of $10^{-2}$ in lifetime of equipment

- What constitutes acceptable risk is domain specific
Risk Assessment

1. Model the system:
   – identify the major components and interfaces
2. Identify hazards & how they arise
   – identify potential failure modes
   – trace consequences and control measures
   – build a cause-and-effect model of the system
3. Analyse and assess risk
   – assess component failure rates
   – assess likelihood & severity of hazards

If some risks are not tolerable, it’s back to the drawing board!
Likelihood of Software Failure?

- Theory of failure-rate prediction is almost non-existent for all but the simplest software
  - same goes for complex hardware, operator procedures, system design, ...
- Design faults now overtaking physical failures in impact on complex systems
- Current best practice relies on the *rigour of the development process* - the **Safety Integrity Level (SIL)**
- Standards differ on exactly what SILs mean, and on what processes are required
  - but broadly speaking, SIL relates to degree to which system safety depends on the component
# IEC 61508: Safety Integrity Levels

In IEC 61508, SILs correspond to acceptable failure rates:

<table>
<thead>
<tr>
<th>SAFETY INTEGRITY LEVEL</th>
<th>DEMAND MODE OF OPERATION (Probability of failure to perform its design function on demand)</th>
<th>CONTINUOUS/HIGH DEMAND MODE OF OPERATION (Probability of a dangerous failure per year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$&gt;=10^{-5} , to &lt; 10^{-4}$</td>
<td>$&gt;=10^{-9} , to &lt; 10^{-8}$</td>
</tr>
<tr>
<td>3</td>
<td>$&gt;=10^{-4} , to &lt; 10^{-3}$</td>
<td>$&gt;=10^{-8} , to &lt; 10^{-7}$</td>
</tr>
<tr>
<td>2</td>
<td>$&gt;=10^{-3} , to &lt; 10^{-2}$</td>
<td>$&gt;=10^{-7} , to &lt; 10^{-6}$</td>
</tr>
<tr>
<td>1</td>
<td>$&gt;=10^{-2} , to &lt; 10^{-1}$</td>
<td>$&gt;=10^{-6} , to &lt; 10^{-5}$</td>
</tr>
</tbody>
</table>
Safety Management

- **Overall goal:** to deliver a safe system, however
  
  “Like justice, safety needs not only to be done, 
  but to be seen to be done.”

- A **Safety Case** documents the claim that the system is 
  safe to be operated

- Main ingredients of a Safety Case:
  - identification of hazards, failure modes, failure 
    mechanisms, safety features, safety targets & SILs
  - reasoned arguments for risk assessment
  - supporting evidence, including: hazard analysis, 
    V&V results
From IEC 61508:

1. Concept
2. Overall scope definition
3. Hazard and risk analysis
4. Overall safety requirements
5. Safety requirements allocation

NOTE 1 Activities relating to verification, management of functional safety and functional safety assessment are not shown for reasons of clarity but are relevant to all overall, E/E/PES and software safety lifecycle phases.

NOTE 2 The phases represented by boxes 10 and 11 are outside the scope of this standard.

NOTE 3 Parts 2 and 3 deal with box 9 (realisation) but they also deal, where relevant, with the programmable electronic (hardware and software) aspects of boxes 13, 14 and 15.
NOTE 1   Activities relating to verification, management of functional safety and functional safety assessment are not shown for reasons of clarity but are relevant to all overall, E/E/PES and software safety lifecycle phases.

NOTE 2   The phases represented by boxes 10 and 11 are outside the scope of this standard.

NOTE 3   Parts 2 and 3 deal with box 9 (realisation) but they also deal, where relevant, with the programmable electronic (hardware and software) aspects of boxes 13, 14 and 15.
Software Engineering for Safety

- All the regular good software-engineering practices
  - thorough requirements analysis, reviews & testing
  - configuration management
- Involve all system stakeholders in safety management
- Design for safety
  - KISS (Keep It Simple, Stupid)
  - no single point of failure
  - isolate critical functions
  - belts *and* braces
  - diversity throughout design, implementation, review
- Pay special attention to internal & external interfaces
Safety-Directed V&V

- **Safety Validation**: are we building a safe system?
  - all hazards & safety requirements identified
  - safety targets are appropriate:
    i.e., if met, will achieve acceptable risk

- **Safety Verification**: are we achieving targets?
  - safety requirements & targets are being flowed down through design
  - appropriate evidence is being gathered that safety targets are being met (and no new hazards introduced)

- Safety Integrity Level determines the *degree of rigour* to be applied
Important Safety V&V techniques

The broad goals of Safety V&V are to

- **identify** (& prioritize) all hazards and
- **trace** their resolution

Different techniques are applicable at different stages of design, according to what design details are available

Will outline 3 techniques that apply well to software:

- Failure Modes & Effects Analysis (FMEA)
- Fault Tree Analysis
- Hazard & Operability Studies (HAZOP)
FMEA Example: Speed Sensor

- Toothed wheel
- Gearbox
- Sensor
- Signal processing unit
- Dashboard
- Gearbox controller
## FMEA Report: Speed Sensor

<table>
<thead>
<tr>
<th>Component</th>
<th>Failure Mode</th>
<th>Local Effect</th>
<th>System Effect</th>
<th>Hazard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed sensor</td>
<td>Breaks</td>
<td>Speed calculated as zero</td>
<td>1. Speedometer shows zero</td>
<td>1. Driver mislead, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. Odometer not updated</td>
<td>2. Maintenance delayed, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. Wrong gear selected</td>
<td>3. Engine seizes at high speed, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Failure Modes and Effects Analysis

- **Method**: from known or predicted failure modes of components, determine possible effects on system

- Good for hazard identification early in development, by considering possible failures of system functions:
  - loss of function (omission failure)
  - function performed incorrectly
  - function performed when not required (commission failure)

- Not so good for multiple failures
Example Fault Tree: tank-level sensors

Tank overflow

- Outlet closed
  - Inlet open
    - Inlet valve failed
      - Controller failed
        - Controller X fails
        - Controller Y fails
  - Outlet closed
    - Outlet valve A
      - Outlet Valve B
    - Controller
      - Controller X
      - Controller Y
Fault Tree Analysis - Summary

- **Method:** trace faults stepwise back through system design to possible causes
  - a tree with a *top event* at the root
  - *logic gates* at branches, linking each event with its "immediate" causes
  - *initiating faults* at leaves (eventually)
- Good for tracing system hazards through to component failures, and thus for allocating safety requirements
- Good for checking completeness of safety requirements
- but can be difficult, time-consuming, hard to maintain
Developed by ICI in mid’60s for hazard identification for chemical process plants

**Method:** given model of the system in terms of “flows” between components
- consider possible deviations in flows, using guide words to steer analysis:
  - no, more, less, as well as, part of, other than, reverse
- consider both causes and effects of deviations

Adapts well as a systematic design-review technique for computer systems (**CHAZOP**)
- guidewords extended with: early, late, before, after
CHAZOP Example - Elevator

Data flow diagram showing internal structure of software

1 Lift panel interface

2 Floor panel interface

3 Sequence controller

Lift request

Movement commands

Floor request

Pending request

Display

Status

Door commands

Control

Feedback

Request

Display

Request

Display

Verification & Validation of Safety Critical Software

SEA’99 Conference
### CHAZOP Example - Elevator Output

<table>
<thead>
<tr>
<th>Interconnection</th>
<th>Attribute</th>
<th>Guide word</th>
<th>Cause</th>
<th>Consequences/implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lift request (Hold-Door-Open)</td>
<td>Data flow</td>
<td>No</td>
<td>Failed button</td>
<td>Sequencer does not receive door hold request. Risk of injury if small/soft item (e.g. scart) caught in door</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Failed wiring</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Failure of lift panel interface</td>
<td></td>
</tr>
<tr>
<td>Lift request (Hold-Door-Open)</td>
<td>Data flow</td>
<td>More</td>
<td>Failure of lift panel interface</td>
<td>Sequence receives spurious door hold request. Doors stay open – lift stuck</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lift request (Hold-Door-Open)</td>
<td>Data flow</td>
<td>Other Than</td>
<td>Equivalent to NO in this context</td>
<td>-</td>
</tr>
</tbody>
</table>
Talk Summary

- Software Safety Engineering is a new discipline
- Standards now require Safety Case prior to operation
- Safety is a system-wide, whole lifecycle issue
- Safety should be designed into a system, rather than added on later
  - start developing safety arguments from earliest stages of design
  - KISS, cost-effectiveness
- Main goals of Safety V&V are to identify all hazards and track their resolution