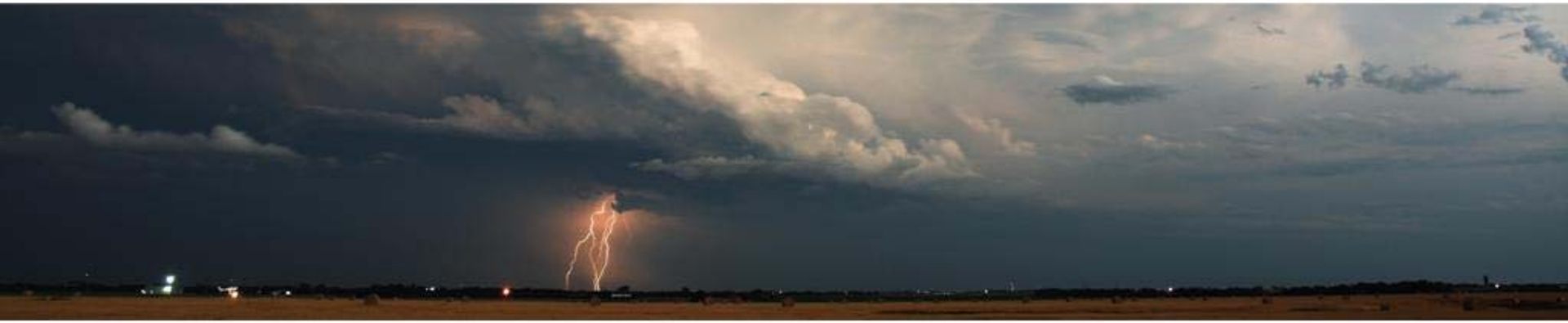
Software Testing and IEC 61508 – Project case study and further thoughts



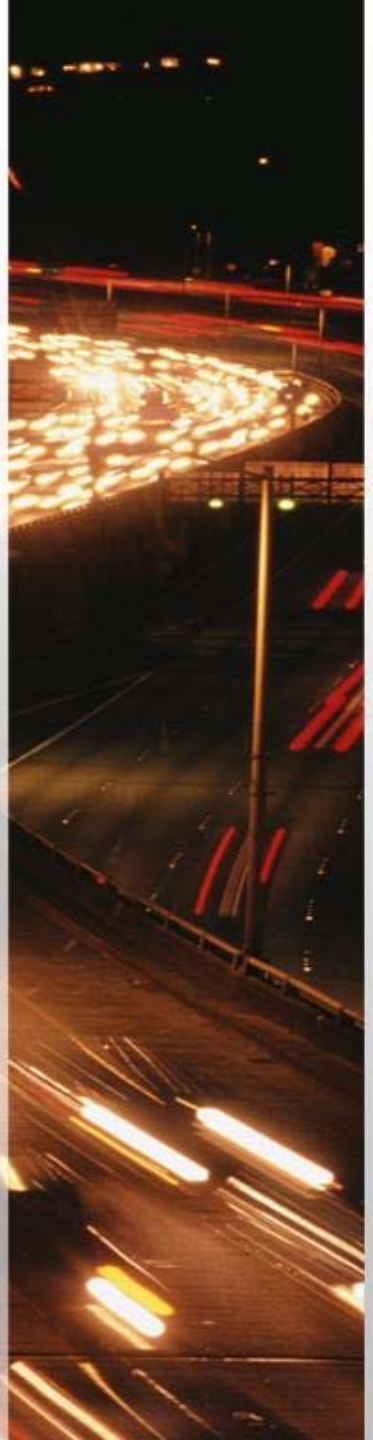
Ian Gilchrist

IPL

Topics

1. Introduction to the NASS Project
2. NASS Testing Strategy
   - Code and Module Testing
   - Integration Testing
   - System Level Testing
3. Conclusions from Project
4. Reflections on IEC 61508 and Testing

Introduction to the NASS project



IPL

UK Motorway Systems

Highways Agency (HA) has overall control

NMCS2 = National Motorway Communications System (2)

- Controls roadside equipment and monitors road conditions

ATM = Active Traffic Management

- Trial program on motorways around Birmingham
- Control traffic according to actual and predicted conditions

NASS = Network ATM Supervisory Subsystem

- An additional element with existing NMCS2 and future ATM systems
- **Predicts congestion, and sets signals and message signs to minimise it**

NASS History

NASS contract for specification, implementation and design was awarded to IPL in 2002

Phase	Delivered	Name	Purpose	Approx code size
A	Jan 2004	Proof of concept	Prove concept	20 KLoC
B	Apr 2005	Demonstrator	Testbed for rules	60 KLoC
C	Mar 2007	NASS V1	Live use	115 KLoC

NASS and IEC 61508 SIL

Highways Agency elected to apply IEC 61508 as the software development/safety standard to be used.

- Some hazards were identifiable

IPL recommended to apply at SIL 1

- NASS does not directly control equipment
- Just issues requests for sign and signal settings
- These may have safety consequences

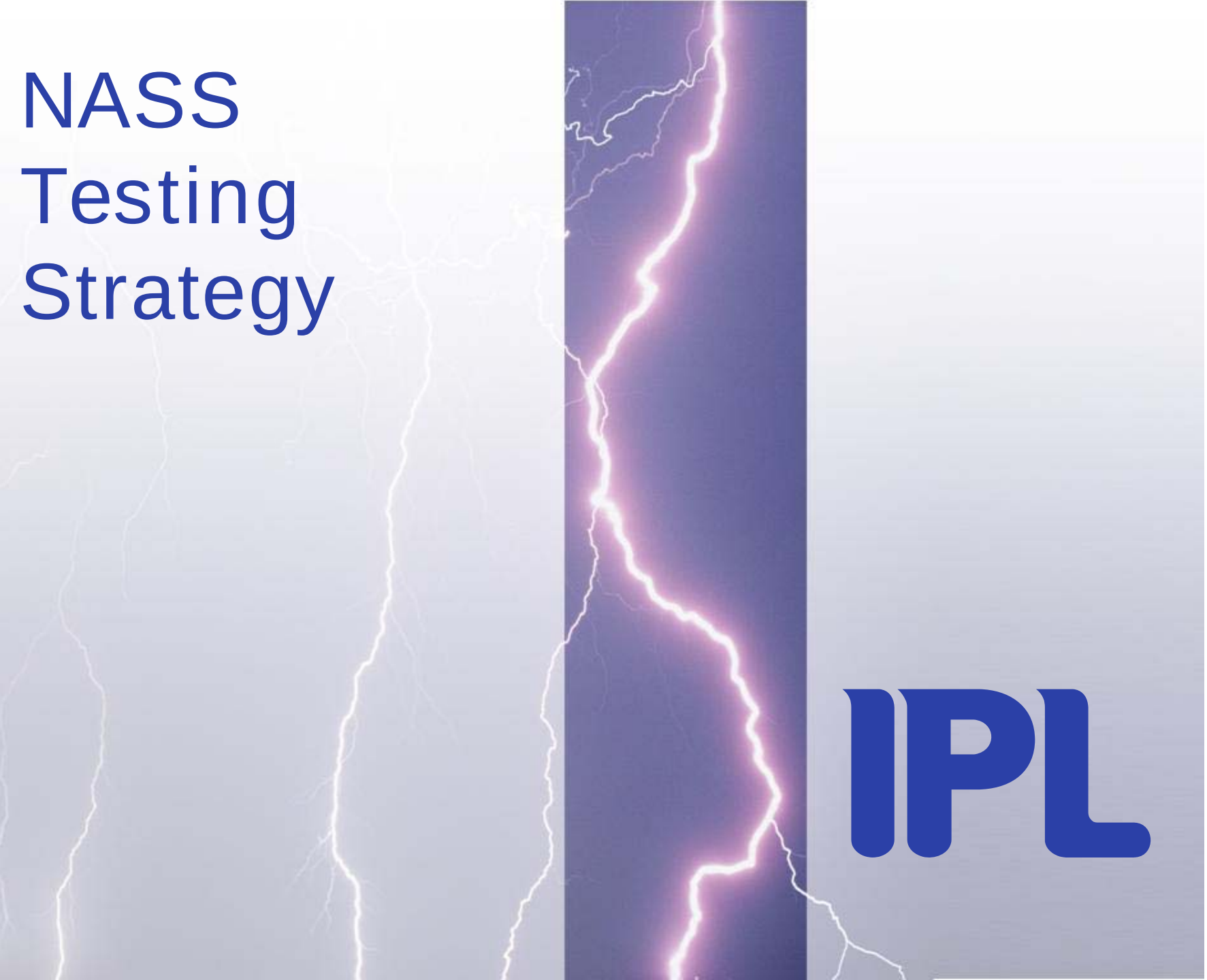
NASS Software Design

Following System Requirements Spec acceptance, used design method based on UML

- Sub-systems (6 in total)
- Components (now 23 in total)
  - Executables
  - (Dynamic Linked) Libraries
- Modules (now 280 in total)
  - C++ classes
    - +Public and Private methods
    - +Code flow shown in pseudo-code

Implementation on Windows, coded with MSVC++ V6

NASS Testing Strategy



IPL

NASS Testing Strategy

HA and IPL project team created a test strategy based on formalised (i.e. independent) testing for

- Each and every entity
- At each every identifiable stage of the project

Each test had its own plan:

- Component and Module tests defined in associated spec
- All higher level tests had own plan in separate document

All tests need to be

- Repeatable
- Self-documenting (results)

NASS Testing Strategy

Design Document	Informs Test Plan for
Existing NMCS documentation	System Interaction Test
NASS System Req’ts Spec	Factory and Site Acceptance tests
Architectural Design Spec	System Integration Tests
Sub-System Design Specs	System Integration Tests
Component Specs	Component tests
C++ Class Specs	Module tests

NASS Module Testing

Each C++ class tested in isolation, using Cantata++

- Stubs and Wrappers used to provide the ‘isolation’ of each class from external interfaces
- Test coverage of 100% Entry-point was mandatory
- Project optionally did coverage to 100% Statement, Decision, and ‘Condition’ coverage (MC/DC definition)
  - Useful additional confidence

NASS Software Integration Testing

After Module testing there were two levels of Software Integration Testing:

- **Component Testing (executables and DLLs)**
  - Also used Cantata++
  - Dummy database for input of data and opportunity to check expected outputs
- **Sub-System Testing**
  - 100% Entry-point coverage (measured with Cantata++?)

NASS System Testing

Several layers of formalised System Tests

- **System Integration Tests**
  - Independently witnessed by IPL QA staff
  - Revealed very few faults
  - Confidence building for next stage:
- **Factory Acceptance Tests**
  - Witnessed by HA staff at IPL offices
  - Took 20 days to run
- **System Interaction Tests**
  - Run at offices of Peek Traffic
  - Demonstrated NASS running smoothly and safely in an exact replica of NMCS2 installation
- **Site Acceptance Tests**
  - Live running of NASS on NMCS2 with simulated data at Regional Control Centre

NASS Testing – all levels

Name	Aimed at	Tool	Comments
Modules	C++ Classes	Cantata++	283 in total
Components	Exes and DLLS	Cantata++	23 in total
[Regression	*All modules and components*	*Cantata++ and IPL framework*	*Run nightly]*
Software Integration	Sub-systems and aspects of System	IPL-developed simulators	
System Integration	System	IPL-developed simulators	Dry-run for FATs
Factory Acceptance	System	IPL-developed simulators + HA 'portable standard'	Witnessed by HA
System Interaction	System	Test rig at Peek	Witnessed by HA
Site Acceptance	System	Live at RCC	Witnessed by HA

NASS Testing

Project Conclusions

IPL

NASS Project Conclusions

Testing served two main purposes

- Developers gained confidence to move from one stage to the next
- Customer (HA) gained confidence that system will work reliably and safely

Other general observations:

- Occupied a large chunk (>50%) of overall project effort
- Testing strategy determined in advance was very useful, with test plans being vital for success
- Test repeatability was very useful
- Test self-documentation was very useful

NASS Project Conclusions

Class test plans were too detailed

- Amount of work disproportionate to benefit gained
- Better concentrate on black-box test planning
- Test classes as whole ‘objects’ better than methods

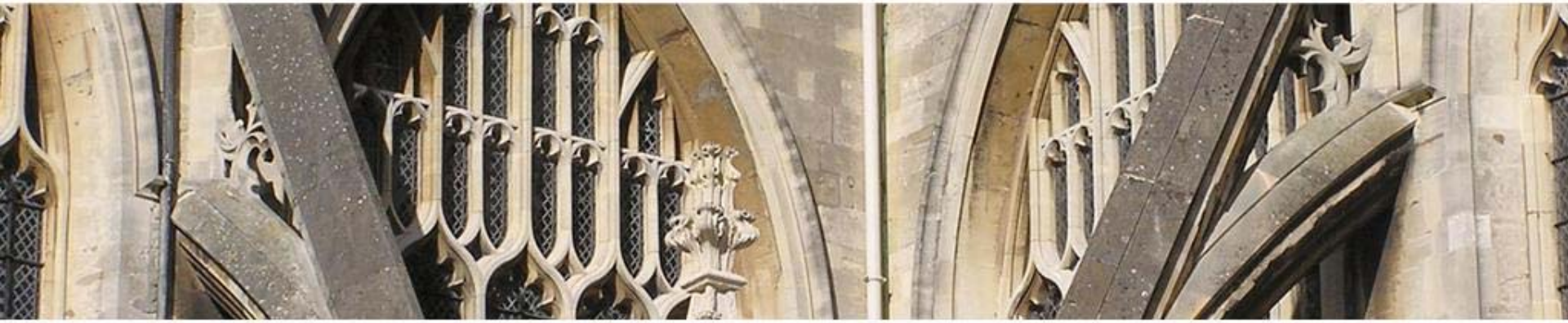
Choice of coverage level was good

- Could have done less than did
- Amount done will easily qualify for SIL 2 if needed

Integration test work was valuable

- Some of the integration tests might have been left out
- In practice were useful at finding faults in low-level design

Reflections on IEC 61508 and Testing Requirements



IPL

IEC 61508 and Testing

The standard was published 1998

- It is now 10 years old
- Based on ideas more than 10 years old
- Is arguably 20 years ‘out of date’

Time to rethink?

- Recommendations still valid and useful?
- Should modern practices be brought in?

Not lose sight of the main aim of testing:

- ‘Performs its intended function’
- ‘Does not perform unintended function’

IEC 61508/3 and Testing

Look at sections:

- 7.4.7 [Software Module Testing]
- 7.4.8 [Software Integration Testing]

And tables:

- A.5 [Software Modules Testing and Integration]
- B.2 [Dynamic Analysis and Testing']
- B.3 [Functional and Black-Box Testing]
- B.6 [Performance Testing]

Refer as needed to 61508/7 [Overview of Techniques and Measures]

Integration Testing

There is not normally much argument about ‘what is Module Testing’

- It was what is done by default as the first level of testing

Integration Testing is a moveable feast

- There may be several layers of Integration testing
- It may be omitted altogether
- What is appropriate in one kind of integration (e.g. software integration) may not be at another (e.g. hardware-software integration

Table A.5 ‘Software Module Testing and Integration’

Technique	Comment
Probabilistic testing (HR at SIL 4)	Software is usually deterministic at the these levels so of dubious relevance.
Dynamic analysis and testing (HR at SILs 2-4)	See comments for Table B.2
Data recording and analysis (HR at all SILs)	‘Recording’ seems merely an aspect of project documentation; why label it as a technique? ‘Analysis’ is undefined except that it ‘may establish a wide variety of information’.
Functional and Black-box testing (HR at all SILs)	This corresponds to what most developers would class as the normal purpose of testing at these levels.
Performance testing (HR at SILs 3-4)	Questionable whether applicable at these levels as it would be rare to specify performance. It would seem more appropriate to shift this item to system level testing rather than keep it here.
Interface testing (HR at SILs 3-4)	As described in Part 7 this would seem an unjustifiable burden on developers.

Table A.5 'Software Module Testing and Integration'

Technique	Comment
Probabilistic testing (HR at SIL 4)	Software is usually deterministic at the these levels so of dubious relevance.
Dynamic analysis and testing (HR at SILs 2-4)	See comments for Table B.2
Data recording and analysis (HR at all SILs)	'Recording' seems merely an aspect of project documentation; why label it as a technique? 'Analysis' is undefined except that it 'may establish a wide variety of information'.
Functional and Black-box testing (HR at all SILs)	This corresponds to what most developers would class as the normal purpose of testing at these levels.
Performance testing (HR at SILs 3-4)	Questionable whether applicable at these levels as it would be rare to specify performance. It would seem more appropriate to shift this item to system level testing rather than keep it here.
Interface testing (HR at SILs 3-4)	As described in Part 7 this would seem an unjustifiable burden on developers.

Table B.2 ‘Dynamic Analysis and Testing’

Technique	Comment
Test case execution from boundary-value analysis (HR for SILs 2-4)	This is a valid activity when combined with equivalence class testing as a means of boosting confidence in software beyond that already achieved with basic functional/structure-based testing.
Test case execution from error guessing (R at all SILs)	Based on the description given in Part 7 this seems a bit random (though based on ‘experience and intuition’).
Test case execution from error seeding (R at SILs 2-4)	This is not really testing but a way of attempting to gauge the effectiveness of existing tests. It is highly arbitrary (who decides what errors to seed), and has little to recommend it.
Performance modelling (HR at SIL 4)	It is unclear why this is included here. Modelling is a validation technique, not testing.
Equivalence classes and input partition testing (HR at SIL 4)	See comment for boundary-value analysis above.
Structure-based testing (HR at SILs 3-4)	This relates to the widely accepted technique of measuring test coverage. See final section.

Table B.2 'Dynamic Analysis and Testing'

Technique	Comment
Test case execution from boundary-value analysis (HR for SILs 2-4)	This is a valid activity when combined with equivalence class testing as a means of boosting confidence in software beyond that already achieved with basic functional/structure-based testing.
Test case execution from error guessing (R at all SILs)	Based on the description given in Part 7 this seems a bit random (though based on 'experience and intuition').
Test case execution from error seeding (R at SILs 2-4)	This is not really testing but a way of attempting to gauge the effectiveness of existing tests. It is highly arbitrary (who decides what errors to seed), and has little to recommend it.
Performance modelling (HR at SIL 4)	It is unclear why this is included here. Modelling is a validation technique, not testing.
Equivalence classes and input partition testing (HR at SIL 4)	See comment for boundary-value analysis above.
Structure-based testing (HR at SILs 3-4)	This relates to the widely accepted technique of measuring test coverage. See final section.

Table B.3 ‘Functional and Black-box Testing’

Technique	Comment
Test case execution from cause-consequence diagrams (R at SILs 3-4)	Applicability depends on the prior production of cause-consequence diagrams. It is not evident that this is a recognised technique in modern terms, though an equivalent might be sought in UML terminology.
Prototyping/animation (R at SILs 3-4)	These techniques see more related to a validation activity than verification/testing.
Boundary-value analysis (HR at SILs 2-4)	It is questionable whether this approach to generating test inputs is a practical proposition at system test level. The combinatorial effect of generating all possible boundary values for an entire system is likely to lead to a collapse of the enterprise under sheer weight of numbers. If a ‘selective’ approach is adopted how does this improve on simple functional testing? This type of testing is best left to the module/integration testing stages.
Equivalence class and input partition testing (HR at SILs 2-3)	Same as above
Process simulation (R at all SILs)	Similar comments as for prototyping/animation as above. Simulation is not a testing technique; it should not be included in this table.

Table B.3 'Functional and Black-box Testing'

Technique	Comment
Test case execution from cause-consequence diagrams (R at SILs 3-4)	Applicability depends on the prior production of cause-consequence diagrams. It is not evident that this is a recognised technique in modern terms, though an equivalent might be sought in UML terminology.
Prototyping/animation (R at SILs 3-4)	These techniques see more related to a validation activity than verification/testing.
Boundary-value analysis (HR at SILs 2-4)	It is questionable whether this approach to generating test inputs is a practical proposition at system test level. The combinatorial effect of generating all possible boundary values for an entire system is likely to lead to a collapse of the enterprise under sheer weight of numbers. If a 'selective' approach is adopted how does this improve on simple functional testing? This type of testing is best left to the module/integration testing stages.
Equivalence class and input partition testing (HR at SILs 2-3)	Same as above
Process simulation (R at all SILs)	Similar comments as for prototyping/animation as above. Simulation is not a testing technique; it should not be included in this table.

Table B.6 ‘Performance Testing’

Technique	Comment
Avalanche/stress testing (HR at SILs 3-4)	Stress testing is a valid and useful technique provided that some performance indicators are laid down in advance
Response timings and memory constraints (HR at all SILs)	These are valid and useful techniques but why have they been combined in one entry? They should arguably be treated as separate requirements
Performance requirements (HR at all SILs)	This is a valid and necessary part of system testing but as with stress testing it does require that performance indicators be defined in advance

Table B.6 ‘Performance Testing’

Technique	Comment
Avalanche/stress testing (HR at SILs 3-4)	Stress testing is a valid and useful technique provided that some performance indicators are laid down in advance
Response timings and memory constraints (HR at all SILs)	These are valid and useful techniques but why have they been combined in one entry? They should arguably be treated as separate requirements
Performance requirements (HR at all SILs)	This is a valid and necessary part of system testing but as with stress testing it does require that performance indicators be defined in advance

Missing from IEC 61508?

IPL

Add to Testing aspects of IEC 61508?

The most obvious deficiency (in the authors' opinion) is the lack of defined test coverage levels (by SIL).

- Test coverage defines 'how much' testing is to be done. This in turn what confidence we can place in the reliable/safety of the code tested.

Defined coverage levels are an aspect of most other safety-related software standards, so why not IEC 61508?

A Suggestion…

From the RTCA DO-178B standard we might adopt the following for measurement of structural coverage:

Coverage Type	SIL 1	SIL 2	SIL 3	SIL 4
100% Entry points	HR	HR	HR	HR
100% Statements	R	HR	HR	HR
100% Decisions	R	R	HR	HR
100% MC/DC	R	R	R	HR

MC/DC?

MC/DC – Modified Condition/Decision Coverage

- Condition/Decision Coverage (C/DC) requires ALL combinations of multiple conditions in a decision to be exercised
- Example: (A and B and C) would require 8 test cases

The ‘modified’ form of C/DC is a reduced form, which claims to achieve the same value but with less effort

- Example: (A and B and C) would require 4 test cases

See Chilenski and Miller, 1994

IEC 61508 and Software Testing

Thank you for listening

Any questions?

ian.gilchrist@ipl.com