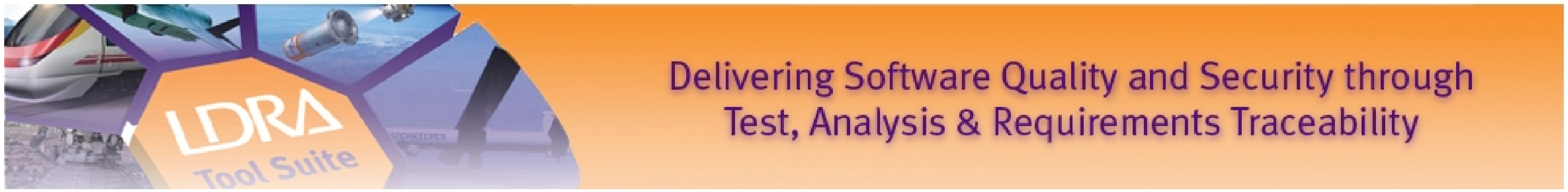
Reduce Medical Device Compliance Costs with Best Practices

mark.pitchford@ldra.com



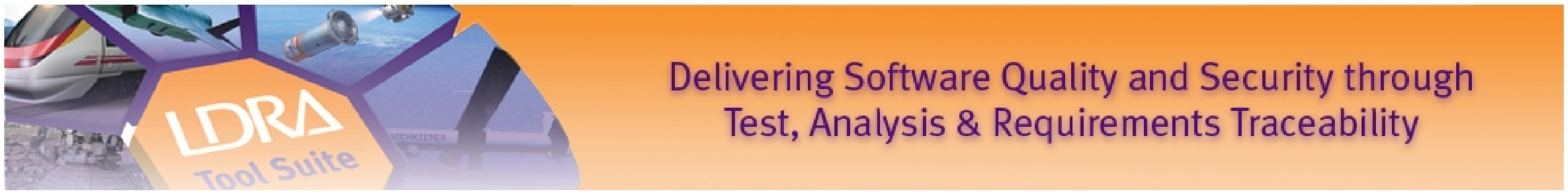
Agenda

- Medical Software Certification
  - How new is Critical Software Certification?
  - What do we need to do?
  - What Best Practises will help us achieve Certification?

- Questions & Answers

CRITICAL SOFTWARE CERTIFICATION

HOW NEW IS IT?



Where is certification enforced?

Whenever the cost of failure is very high

Risk of death or injury

High cost of repair

High cost of product recall

What software needs to be certified?

Aircraft

Trains

Medical Devices

Nuclear Power Stations

Cars

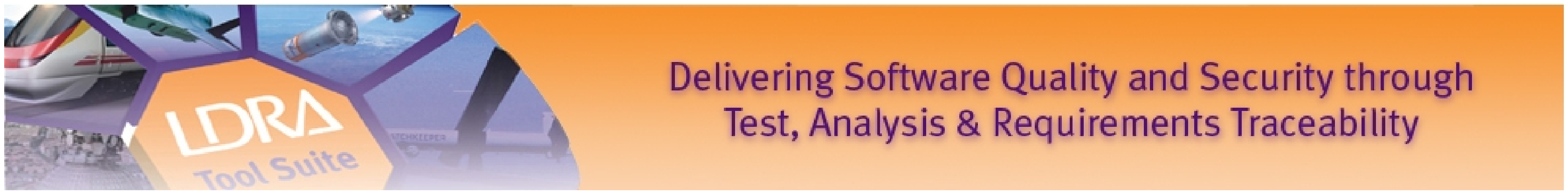
Industrial Plants

Leading Safety Critical Standards

Avionics	DO-178B (First published 1992) / DO-178C
Industrial	IEC 61508 (First published 1998)
Railway	CENELEC EN 50128 (First published 2001)
Nuclear	IEC 61513 (First published 2001)
Automotive	ISO/DIS 26262 (Draft)
Medical	IEC 62304 (First published 2006)
Process	IEC 61511 (First published 2003)

So, the experience of other sectors is invaluable to the medical device (and automotive) industries.

IEC 62304 AND RELATED IEC 61508 DERIVATIVES



Safety Integrity Levels

IEC 61508 (Industrial)

SIL Level 1 to 4

ISO/DIS 26262 (Automotive)

ASIL A to ASIL D

IEC 62304 (Medical)

Class A to Class C

CENELEC EN 50128 (Railway)

SIL Level 0 to SIL Level 4

DO-178B / DO-178C (Avionics)

Level E to Level A

So, nothing new here either!

Functional Safety Assessment

Classes A – C in IEC 62304 are based on the principle of IEC 61508's SIL levels ...

Minimum Level of Independence	Safety Integrity Level			
	1	2	3	4
Independent Person	HR	HR	NR	NR
Independent Department	-	HR	HR	NR
Independent Organization	-	-	HR	HR

Table 2: Assessment independence level for E/E/PE and software life cycle activities

(E/E/PE) : Electrical / Electronic / Programmable Electronic systems

IEC 61508 and IEC 62304

LDRA

IEC 61508 based standards are

- Primarily process oriented
- Includes Verification and Validation(V&V) guidelines for that process

IEC 61508 based standards define the need for

- Software requirements
- The safety lifecycle for software,
- Validation and verification appropriate for each SIL (or class)

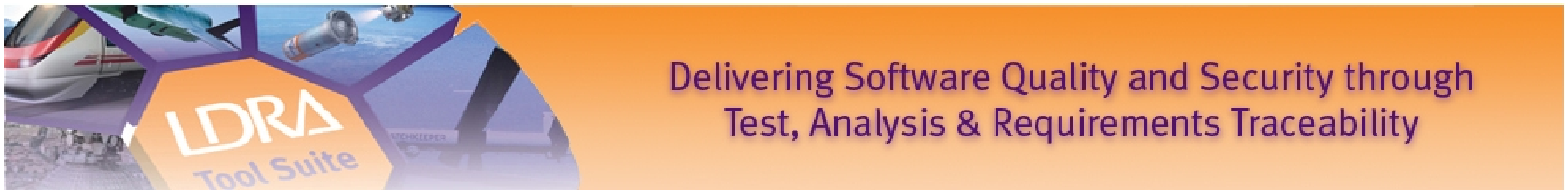
IEC 61508 based standards require V&V activities including:

- Verification of code
- Software module testing
- Software integration testing

Best practises to achieve these aims are long established elsewhere and so can easily be adopted by the medical devices industry in meeting IEC 62304

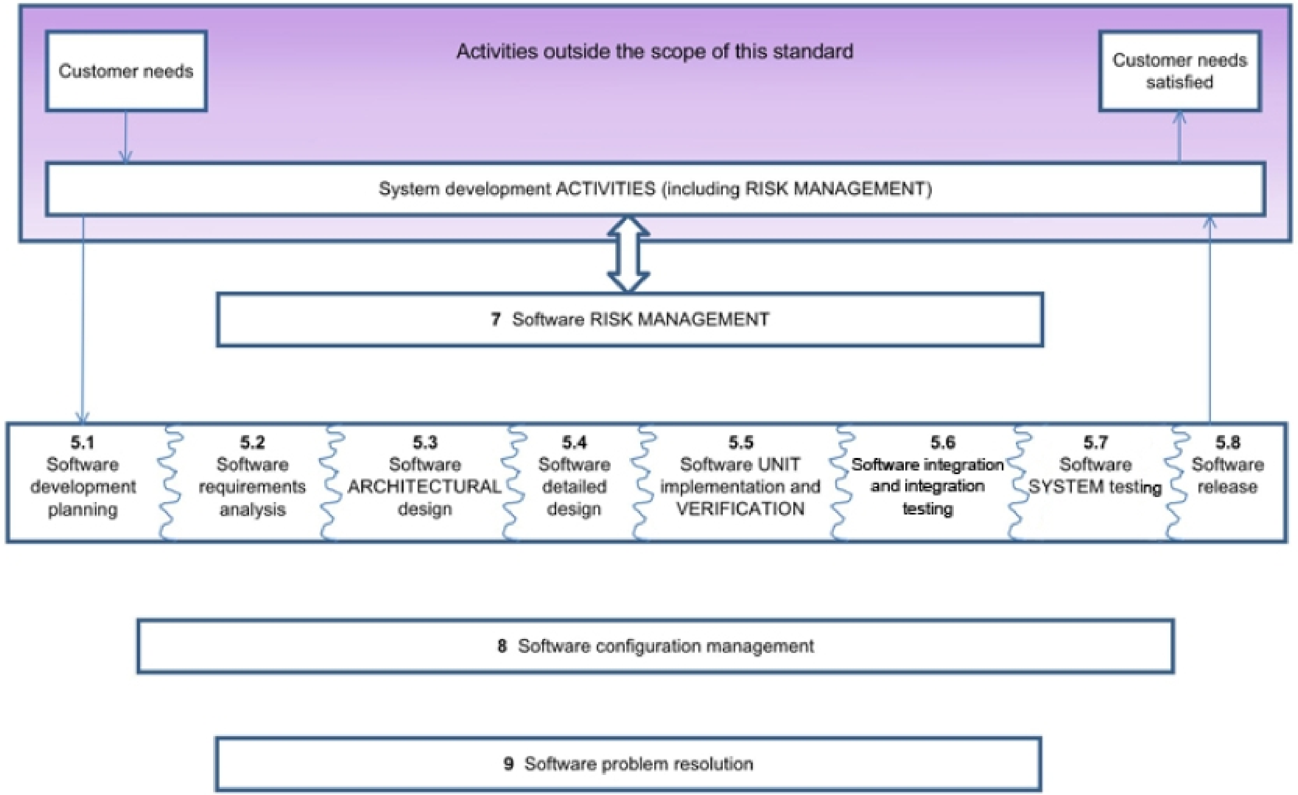
CRITICAL SOFTWARE CERTIFICATION

WHAT DO WE NEED TO DO?



IEC 62304 : Common Framework

- The set of processes, activities, and tasks described in this standard establishes a common framework for medical device software life cycle processes



IEC62304 : Clause 5

- IEC 62304 Clause 5 details the software development process of the product. It specifically addresses:

	Process
5.1	Software development planning
5.2	Software requirements analysis
5.3	Software architectural design
5.4	Software detailed design
5.5	Software unit implementation and verification
5.6	Software integration and integration testing
5.7	Software system testing
5.8	Software release

IEC62304 : Clause 5.3

- IEC 62304 Clause 5.3 details the Software architectural design:

	5.3 Software architectural design
5.3.1	Transform software requirements into an ARCHITECTURE
5.3.2	Develop an ARCHITECTURE for the interfaces of SOFTWARE ITEMS
5.3.3	Specify functional and performance requirements of SOUP item
5.3.4	Specify SYSTEM hardware and software required by SOUP item
5.3.5	Identify segregation necessary for RISK CONTROL
5.3.6	Verify software ARCHITECTURE

SOUP = Software Of Unknown Pedigree

Safety Integrity Levels

- The IEC 62304 standard expects the manufacturer to assign a safety class to the software system as a whole
- This classification is based on the potential to create a hazard that could result in an injury to the user, the patient or other people
- There are three software classes:

Class	Failure Impact
A	No injury or damage to health is possible
B	Non serious injury is possible
C	Death or serious injury is possible

Impact of Software Safety Classification

- The safety classification has a significant impact on the software development life cycle

Software Documentation		Class A	Class B	Class C
Clause	Subclauses			
Software development plan	5.1.1, 5.1.2, 5.1.3, 5.1.6, 5.1.7, 5.1.8, 5.1.9	X	X	X
	5.1.5, 5.1.10, 5.1.11		X	X
	5.1.4			X
Software requirements specification	5.2.1, 5.2.2, 5.2.4, 5.2.5, 5.2.6	X	X	X
	5.2.3		X	X
Software architecture	5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.3.6		X	X
	5.3.5			X
Software detailed design	5.4.1		X	X
	5.4.2, 5.4.3, 5.4.4			X
Software unit implementation and verification	5.5.1	X	X	X
	5.5.2, 5.5.3, 5.5.5		X	X
	5.5.4			X
Software integration and integration testing	All requirements		X	X
Software system testing	All requirements		X	X
Software release	5.8.4	X	X	X
	5.8.1, 5.8.2, 5.8.3, 5.8.5, 5.8.6, 5.8.7, 5.8.8		X	X
Software maintenance process	6.1, 6.2.1, 6.2.2, 6.2.4, 6.2.5	X	X	X
	6.2.3		X	X
Software risk management process	7.1, 7.2, 7.3, 7.4.2, 7.4.3		X	X
	7.4.1	X	X	X

MEDICAL SOFTWARE CERTIFICATION

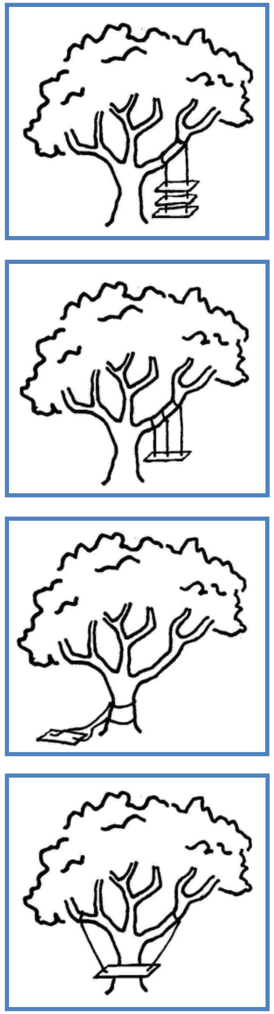
WHAT BEST PRACTICES SHOULD WE APPLY?

Recommended Best Practices

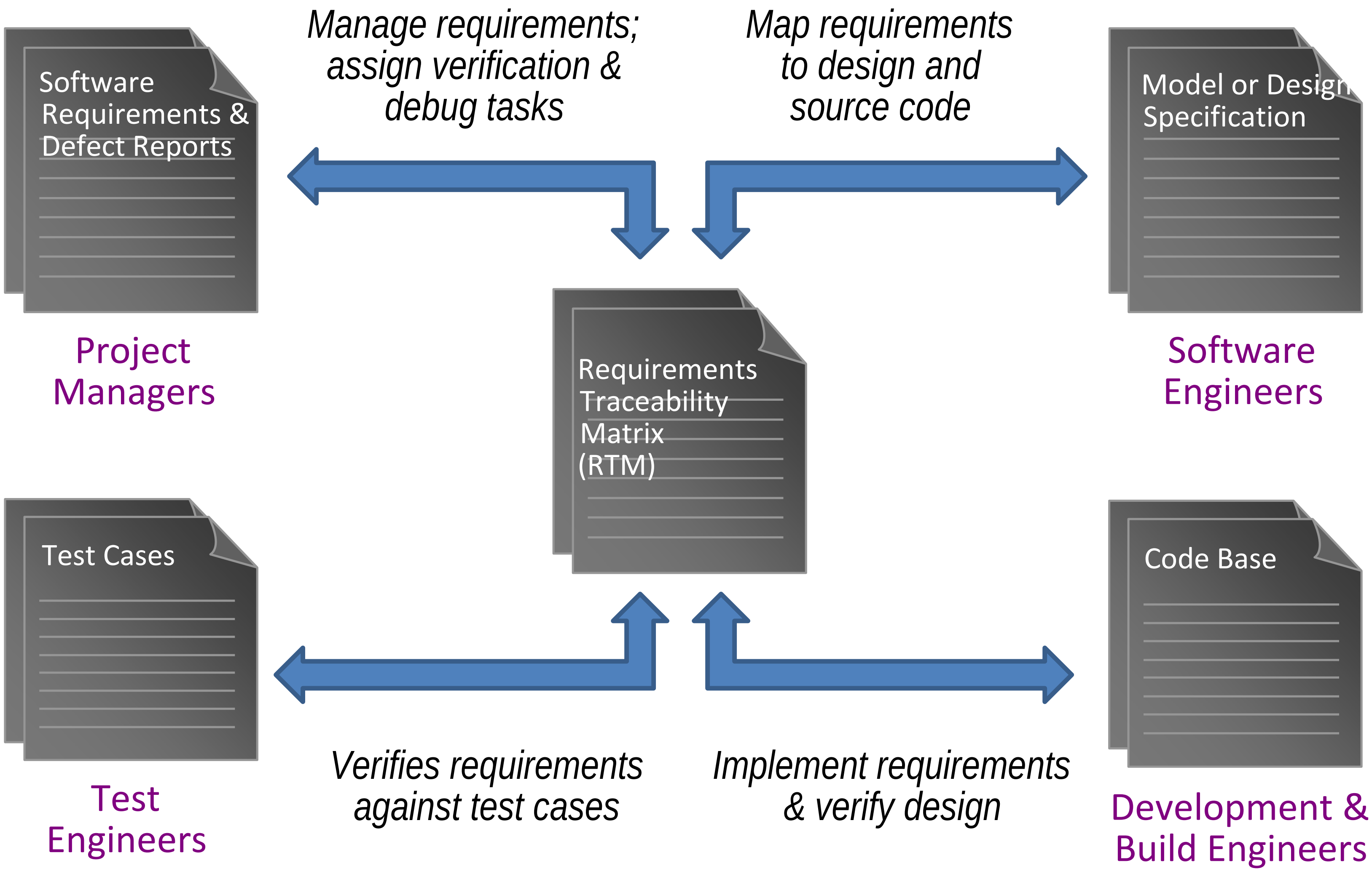
- Requirements
  - Trace Requirements
- Static Analysis
  - Coding Standard
  - Check Complexity
  - Control Flow Analysis
  - Data Flow Analysis
- Dynamic Analysis & Unit Testing
  - Structural Coverage
- Test independence

Avoid the Requirement Gap

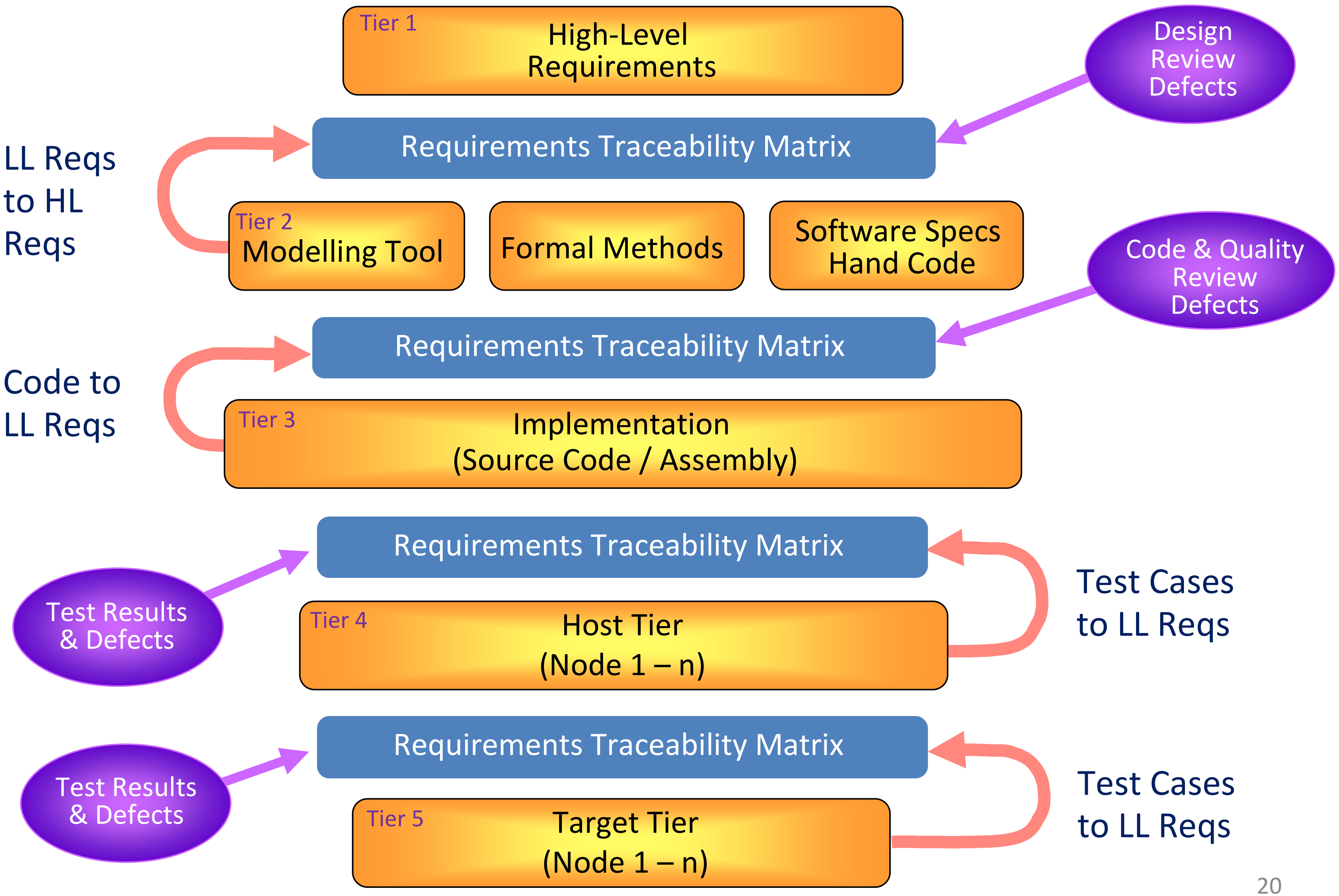
- Process must be "right weight"
  - Not too heavy, not too light
  - Help rather than hinder
  - No bias to particular disciplines or phases
- Focus on requirements
  - Don't ignore them once construction begins
  - Implement what the stakeholder wants
- Manage requirements
  - Continually refine
  - Apply quality criteria
- Trace requirements



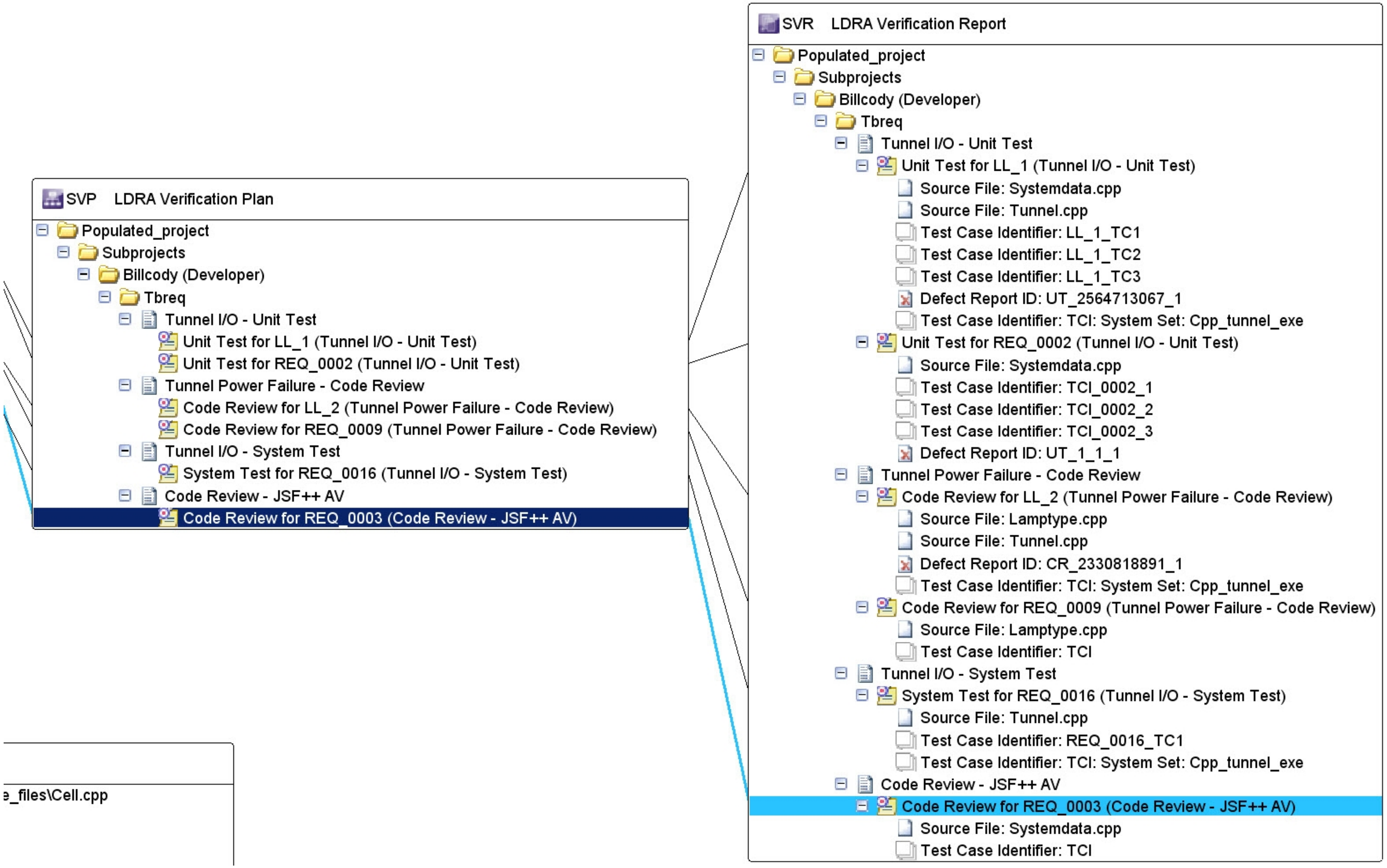
Requirements Drive Development



Traceability Across Development Tiers



Requirements Traceability



Requirements Traceability - Minimizing the overhead

- Traditionally a labour intensive process
  - even if static & dynamic analysis are automated.

- Automation improves quality and reduces costs through
  - Less room for human error
  - Automatic analysis of the “knock on” effects of changes
  - Reference point when changes are requested
  - A maintained RTM even when the pressure is on

STATIC ANALYSIS

Coding Standards for new developments

Quality

The best way to avoid having defects in the code is not to put them in

Roughly 80% of C/C++ software defects are attributable to issues with 20% of the language constructs

Standards such as MISRA-C:2004 and MISRA C++:2008 avoid this subset to improve quality

Security

Standards such as Cert C avoid language constructs that can lead to exploitable vulnerabilities

Style

Ensure that code is written in a particular style

Coding Standards - Minimizing the overhead

Automating "peer review" improves quality and reduces costs through

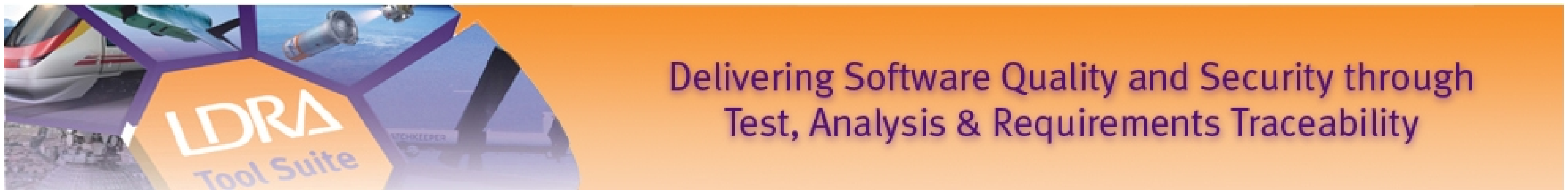
- Consistency of interpretation
- Consistency of application (no "Friday afternoon" effect!)
- Removal of potential for tension between participants.
- Speed of review process

When this is a new concept, initial resistance is likely...

But it soon becomes a learning tool...

... And ultimately merely confirms that the standard is being met

COMPLEXITY ANALYSIS



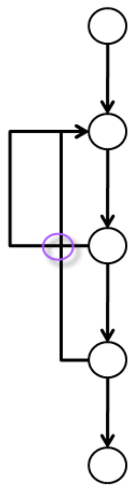
Why use Complexity Metrics?

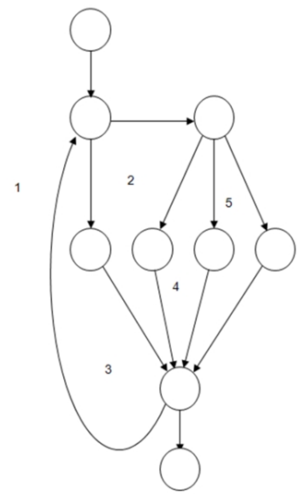
- Code is sometimes complicated.
- Sometimes complicated code is addressing a complex problem.
  - That is unavoidable!
- Sometimes complicated code is not addressing a complex problem. That code:
  - Is prone to costly error at the point of release
  - Is prone to costly error during modification
  - Will demand disproportionately extensive tests whenever changes are made

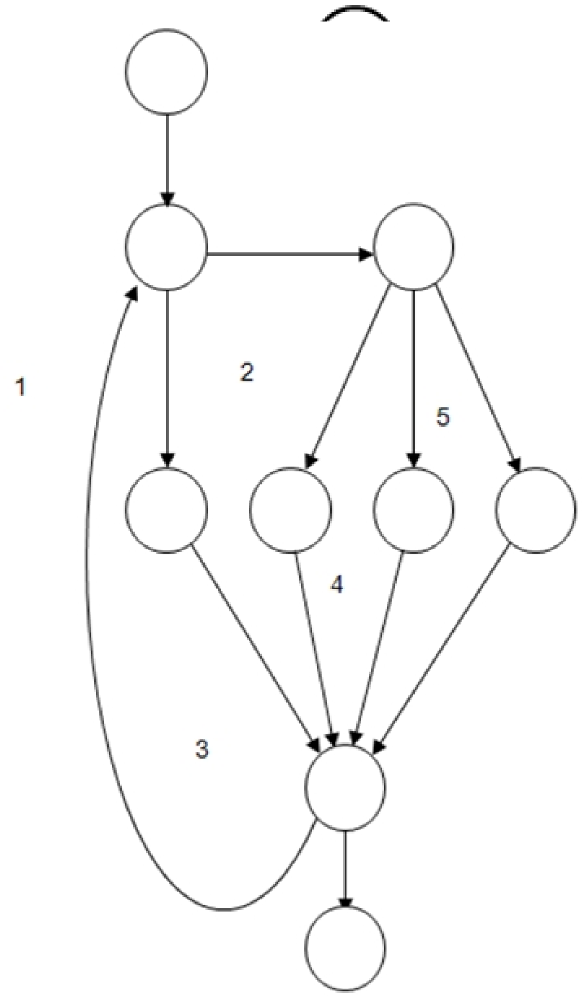
Complexity Metrics

- The principal Complexity Metrics are:
  - Knots
  - Cyclomatic Complexity

- Additional complexity metrics are:
  - Essential Knots
  - Essential Cyclomatic Complexity







Complexity Analysis - Minimizing the overhead

- Sometimes complicated code is not addressing a complex problem. That code:
  - Is prone to costly e poi e
  - Is prone to costly erro on
  - Will demand disproporti sive tests whenever changes are made

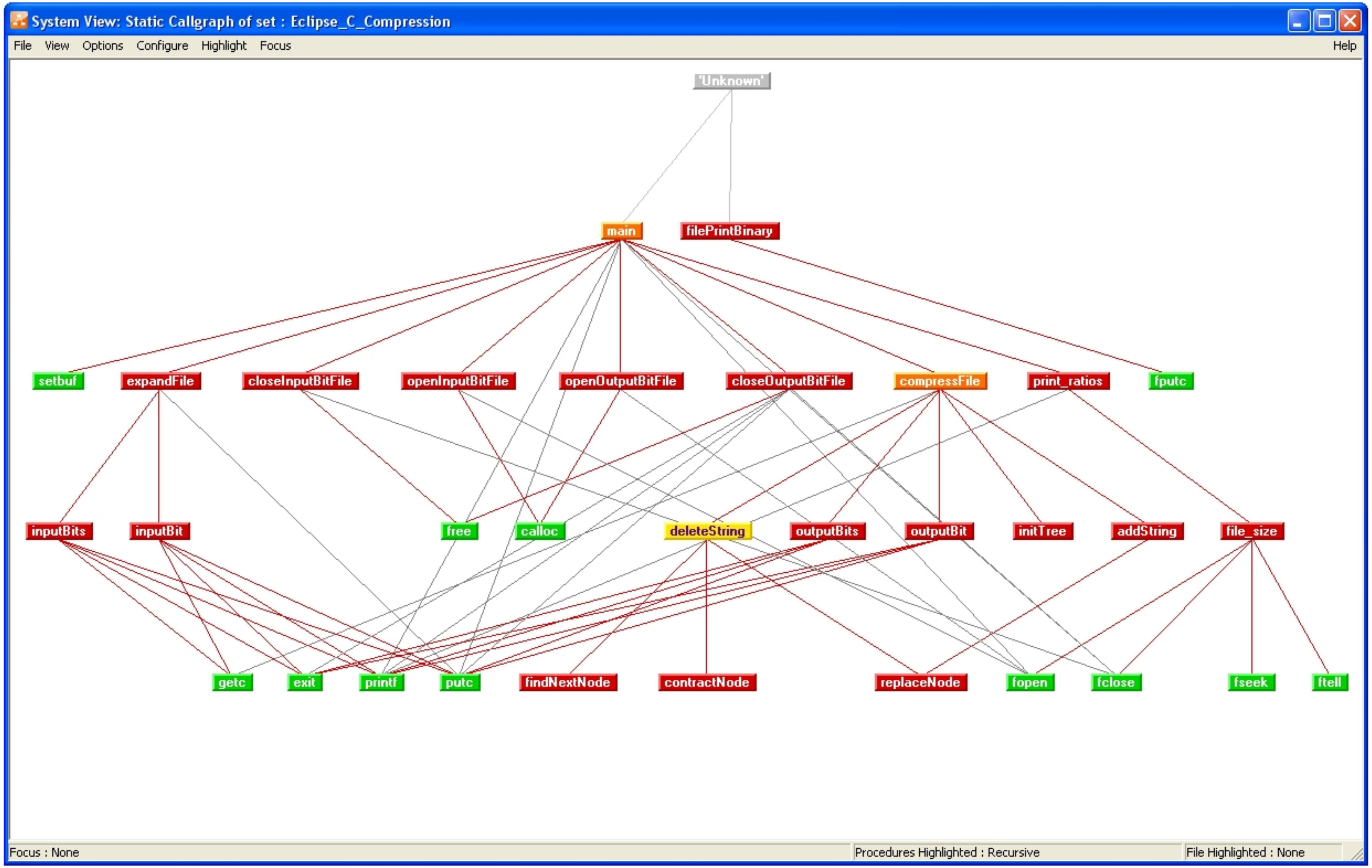
Control Flow & Data Flow Analysis

- Control flow analysis
  - Control Flow Analysis is performed both on the program calling hierarchy and on the individual procedures. The rules of structured programming are applied and defects reported
- Static data flow analysis
  - Follows variables through the source code and reports any anomalous use. This is performed at procedure level and also as part of the system wide analysis

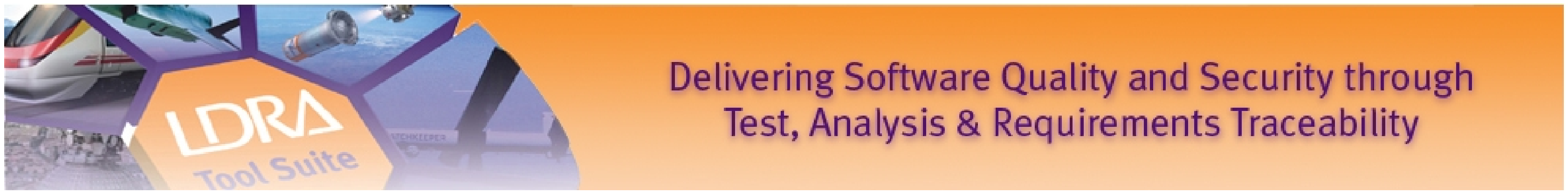
Technique/Measure	Ref	SIL1	SIL2	SIL3	SIL4
1 Boundary value analysis	C.5.4	R	R	HR	HR
2 Checklists	B.2.5	R	R	R	R
3 Control flow analysis	C.5.9	R	HR	HR	HR
4 Data flow analysis	C.5.10	R	HR	HR	HR
5 Error guessing	C.5.5	R	R	R	R
6 Fagan inspections	C.5.15	---	R	R	HR
7 Sneak circuit analysis	C.5.11	---	---	R	R
8 Symbolic execution	C.5.12	R	R	HR	HR
9 Walk-throughs/design reviews	C.5.16	HR	HR	HR	HR

Table B.8 – Static analysis

Call Graph : Control Flow Visualisation



DYNAMIC ANALYSIS – UNIT TEST & CODE COVERAGE

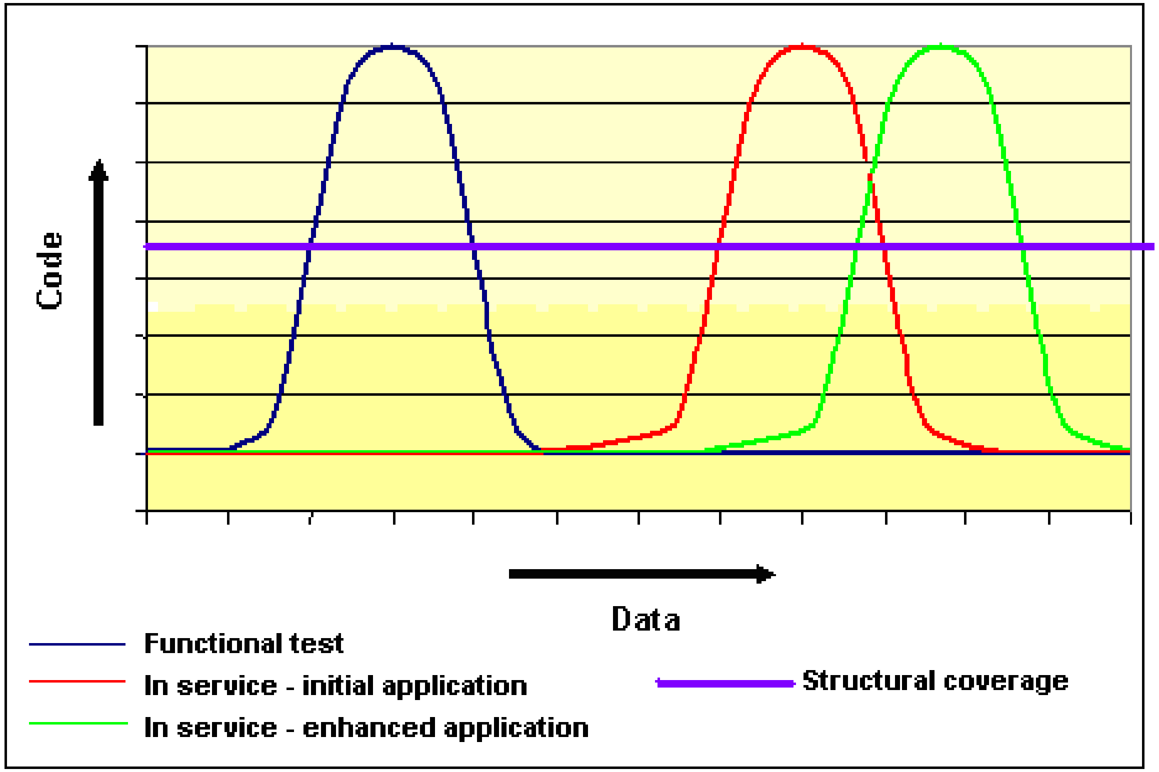


Why use Unit/Module Test?

- **Unit testing** focuses on the behaviour of execution of a subset of application code.
  - Code is compiled and executed in a similar environment to that used by the application under development
- Unit testing traditionally employs a bottom-up testing strategy in which units are tested and then integrated with other test units.
- There is clearly no complete code set to hand to initiate tests such as these, which implies the need for “harness” code to allow the code to build.

Why use Structural Coverage?

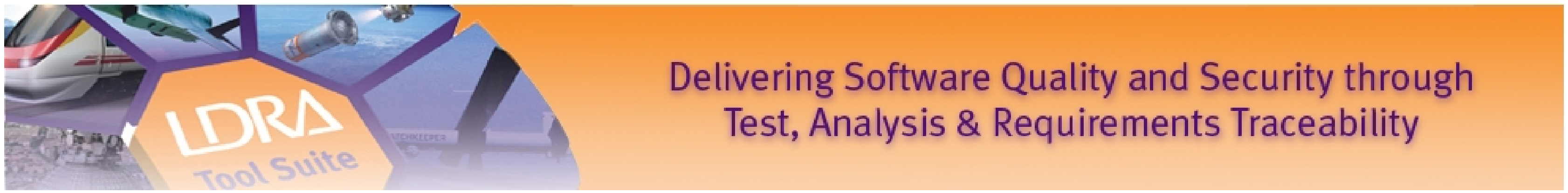
- Consistent coverage produces software ready for all eventualities.
- Code coverage data from Unit and System testing can be combined



Unit Testing and Code Coverage - Minimizing the overhead

- Automated Unit test tools are designed to automatically generate the harness code.
  - This means that tests focus on the application code, and there is no need to debug the harness code itself!
- Unit test sequences can be stored and re-executed at will, from batch files if desired.
- Code Coverage from Unit Test or System Test can be used in isolation or combination.
- The “test-modify-retest” process cycle can be undertaken even under version control.

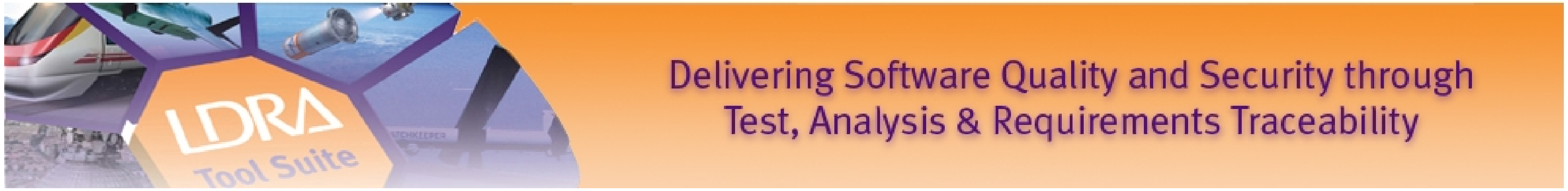
TEST TOOLS & TEST INDEPENDENCE



Test tools and test independence

- Static analysis
  - The interpretation of coding rules is consistent and repeatable.
- Unit test
  - Ideally, dynamic tests should be carried out independently.
  - Where that is not practical, test tools provide a framework which itself lends an element of independence.
  - Traditional Unit testing demands a certain knowledge of the code in order to write the harness.
  - Robustness tests through automatic vector generation.
- System test
  - Code coverage confirms the extent to which code has been exercised.

SUMMARY


LDRA
Tool Suite
Delivering Software Quality and Security through
Test, Analysis & Requirements Traceability

Summary

LDRA

- How new is Software Standards Compliance?
- What do we need to do?
  - IEC 62304 and other standards
  - Class levels
- How do we apply best practice?
  - Requirements Traceability
  - Coding Standards
  - Control Flow and Data Flow Analysis
  - Software Module Testing
  - Structural Coverage & Unit Test
  - Test tools & test independence

For further information:

www.ldra.com

mark.pitchford@ldra.com

jonathan.kelly@ldra.com