ISO 26262 the Emerging Automotive Safety Standard

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

Agenda

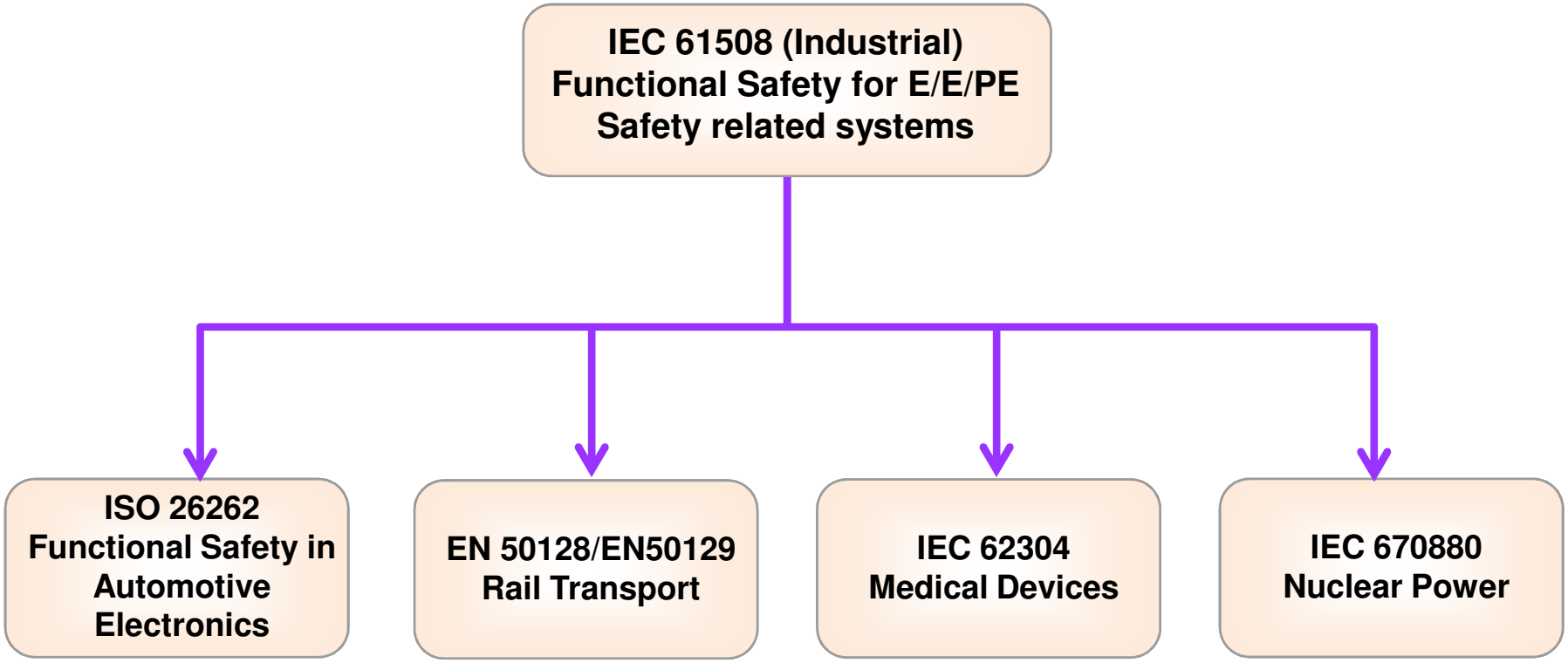
- Introduction of ISO/DIS 26262 (ISO 26262)
- Parts of ISO 26262
- ASIL Levels
- Part 4 : Product Development – System Level
- Part 6 : Product Development – Software Level
- Fitting software tools into ISO 26262 process

Introduction of ISO/DIS 26262 (ISO 26262)

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

Introduction

ISO 26262 is an adaptation of IEC 61508 for the automotive industry.



Introduction

- It applies to safety-related road vehicle E/E systems, and addresses hazards due to malfunctions, excluding nominal performances of active and passive safety systems.
- Risk is determined based on customer risk by identifying the so-called Automotive Safety Integrity Level (ASIL) associated with each undesired effects.
- It provides ASIL-dependent requirements for the whole lifecycle of the E/E system (including Hardware and Software components)

Parts of ISO 26262

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

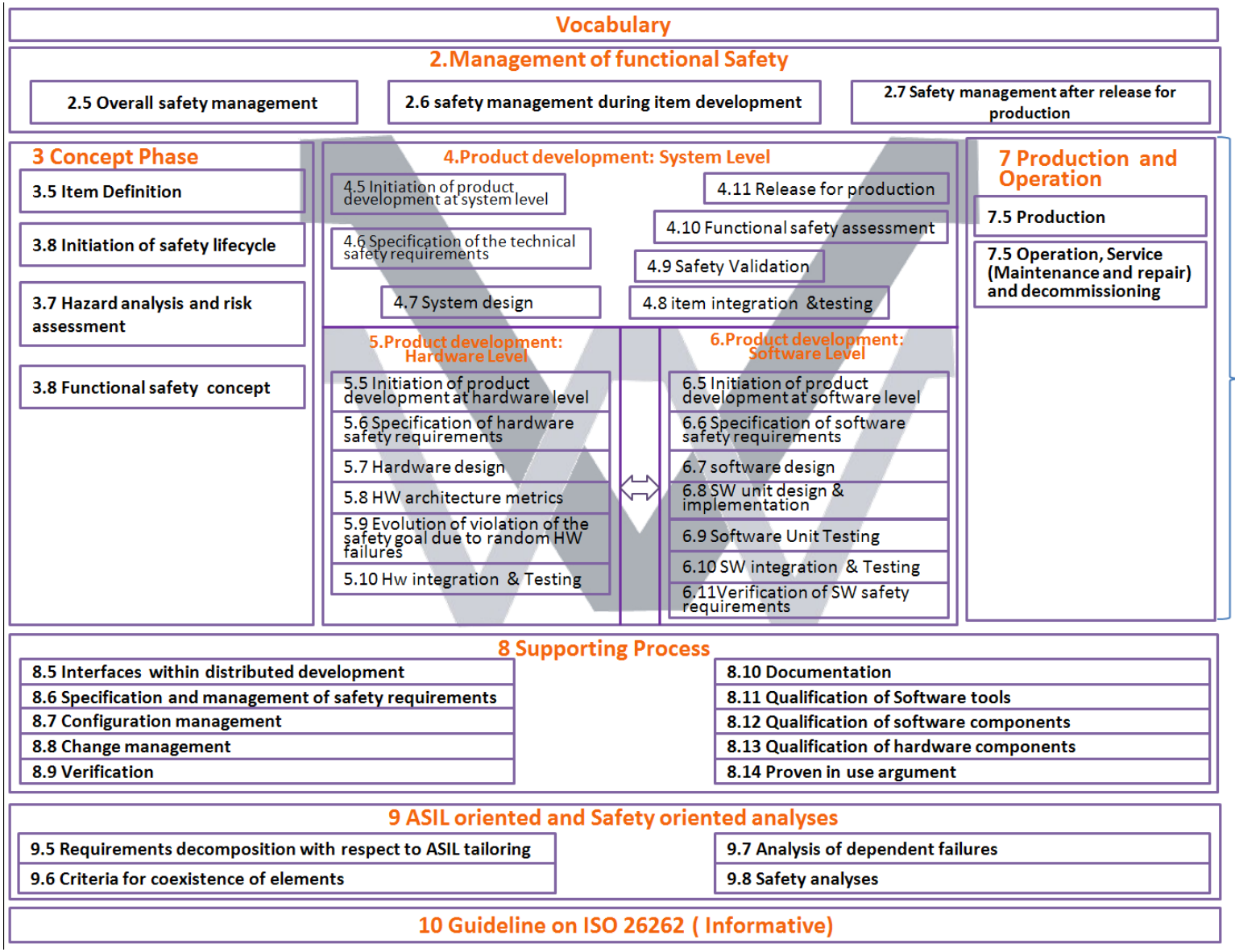
ISO 26262 Parts

- Part 1 : Vocabulary
- Part 2 : Management of Functional Safety
- Part 3 : Concept phase
- Part 4 : Product Development: System Level
- Part 5 : Product Development: Hardware Level
- Part 6 : Product Development: Software Level
- Part 7 : Production and Operation
- Part 8 : Supporting Processes
- Part 9 : ASIL-oriented and Safety-oriented Analyses
- Part 10 : Guidelines on ISO 26262

IEC 61508 versus ISO 26262

IEC 61508	ISO/DIS 26262
Part 1: General requirements	**Part 1**: Vocabulary
Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems	**Part 2**: Management of functional safety
Part 3: Software requirements	**Part 3**: Concept phase
Part 4: Definitions and abbreviations	**Part 4**: Product Development: System Level
Part 5: Examples of methods for the determination of safety integrity levels	**Part 5**: Product Development: Hardware Level
Part 6: Guidelines on the application of parts 2 and 3	**Part 6**: Product Development: Software Level
Part 7: Overview of techniques and measures	**Part 7**: Production and Operation
	Part 8: Supporting Processes
	Part 9: ASIL-orientated and safety-oriented analysis
	Part 10: Guideline

ISO 26262 Process



ASIL Levels

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

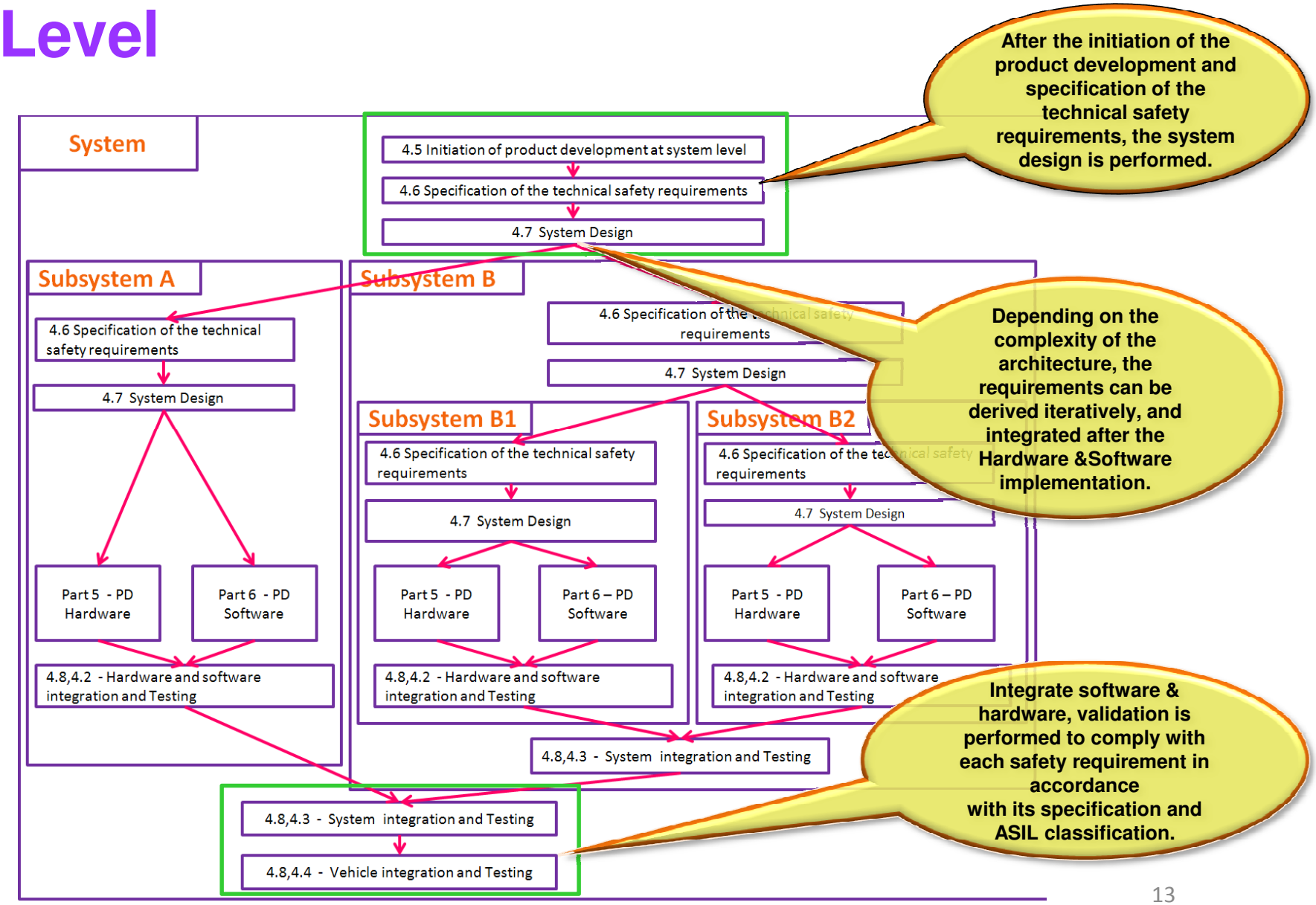
ASIL

- ISO/DIS 26262 replaces SILs with ASILs (Automotive Safety Integrity Levels)
- ASILs designed to specify the measures required to avoid unreasonable residual risk
- ASIL levels A-D, with D being the most demanding
- Risk of each hazardous event is evaluated on the basis of:
  - Frequency of the situation (or “exposure”)
  - Impact of possible damage (or “severity”)
  - Controllability

Part 4 : Product Development – System Level

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

Part 4 : Product Development – System Level

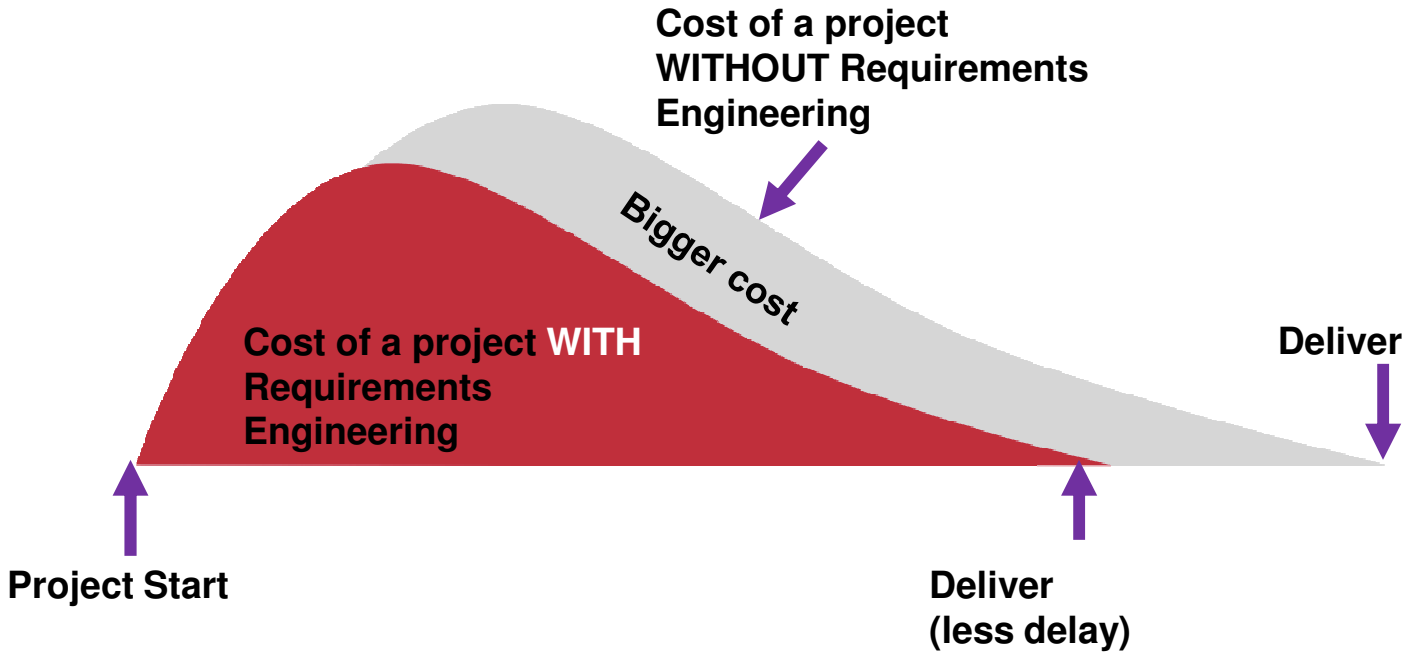


4.6 - Specification of the Technical Safety Requirements

- Objective is to develop the technical safety requirements, which refine the functional safety concept considering the preliminary architectural design.
- To verify through analysis that technical safety requirements comply to the functional safety requirements.
- To bring item-level functional safety requirements into system-level technical safety requirements, down to the allocation to hardware and software elements

4.6 - Specification of the Technical Safety Requirements

Requirements Engineering of this nature requires additional initial effort but benefits will be obtained whenever changes are required.



4.7 - System Design

- To develop the system design and the technical safety concept that comply with the functional requirements and the technical safety requirements specification.
- Verify the System design and technical safety concept comply with Technical safety requirements specification.
- Need to have bidirectional traceability between System design and Technical safety requirements specification.

4.8 - Item integration and testing

- To integrate the different parts that compose the system, included other technologies and/or external entities, and to test the obtained product to comply with each safety requirement and to verify that the design has been correctly implemented.

- The integration and testing is carried out from software-hardware integration, and going through integration of systems up to vehicle integration, with specific tests performed at each integration phase.

4.8 - Item integration and testing

- Each functional and technical safety requirements shall be tested at least once in the complete integration phase.

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Analysis of external and internal interfaces	+	++	++	++
1c	Generation and analysis of equivalence classes for hardware software integration	+	+	++	++
1d	Analysis of boundary values	+	+	++	++
1e	Knowledge or experience based error guessing	+	+	++	++
1f	Analysis of functional dependencies	+	+	++	++
1g	Analysis of common limit conditions, sequences, and sources of common cause	+	+	++	++
1h	Analysis of environmental conditions and operational use cases	+	++	++	++
1i	Analysis of field experience	+	++	++	++

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Fault injection test[b]	+	++	++	++
1c	Back-to-back test[c]	+	+	++	++

[a] A requirements-based test denotes a test against functional and non-functional requirements.

[b] A fault injection test uses special means to introduce faults into the test object during runtime. This can be done within the software via a special test interface or specially prepared hardware. The method is often used to improve the test coverage of the safety requirements, because during normal operation safety mechanisms are not invoked.

[c] A back-to-back test compares the responses of the test object with the responses of a simulation model to the same stimuli, to detect differences between the behaviour of the model and its implementation.

Methods		ASIL			
		A	B	C	D
1a	Back-to-back test[a]	+	+	++	++
1b	Performance test[b]	+	++	++	++

[a] A back-to-back test compares the responses of the test object with the responses of a simulation model to the same stimuli, to detect differences between the behaviour of the model and its implementation.

[b] A performance test can verify the performance concerning e.g. task scheme, timing, power output in the context of the whole test object, and can verify the ability of the hardware to run with the intended control software.

”++” The method is highly recommended for this ASIL.

“+“ The method is recommended for this ASIL.

“o“ The method has no recommendation for or against its usage for this ASIL.

4.9 - Safety Validation

- To provide evidence of due compliance with the functional safety goals and that the safety concepts are appropriate for the functional safety of the item.
- To provide evidence that the safety goals are correct, complete and fully achieved at vehicle level.
- The validation plan shall include:
  - 1. The configuration of the item
  - 2. The specification of test cases and acceptance criteria
  - 3. The required environmental conditions

4.10 - Functional safety assessment

- To assess the functional safety that is achieved by the item.

4.11 – Release for Production

- To specify the criteria for the release for production at the completion of the item development.
- The release for production confirms that the item complies with the requirements for functional safety at vehicle level.
- The documentation shall include
  - a) the name and signature of the person in charge of release
  - b) the version/s of the released item
  - c) the configuration of the released item
  - d) references to associated documents
  - e) the release date

Part 6 : Product Development – Software Level

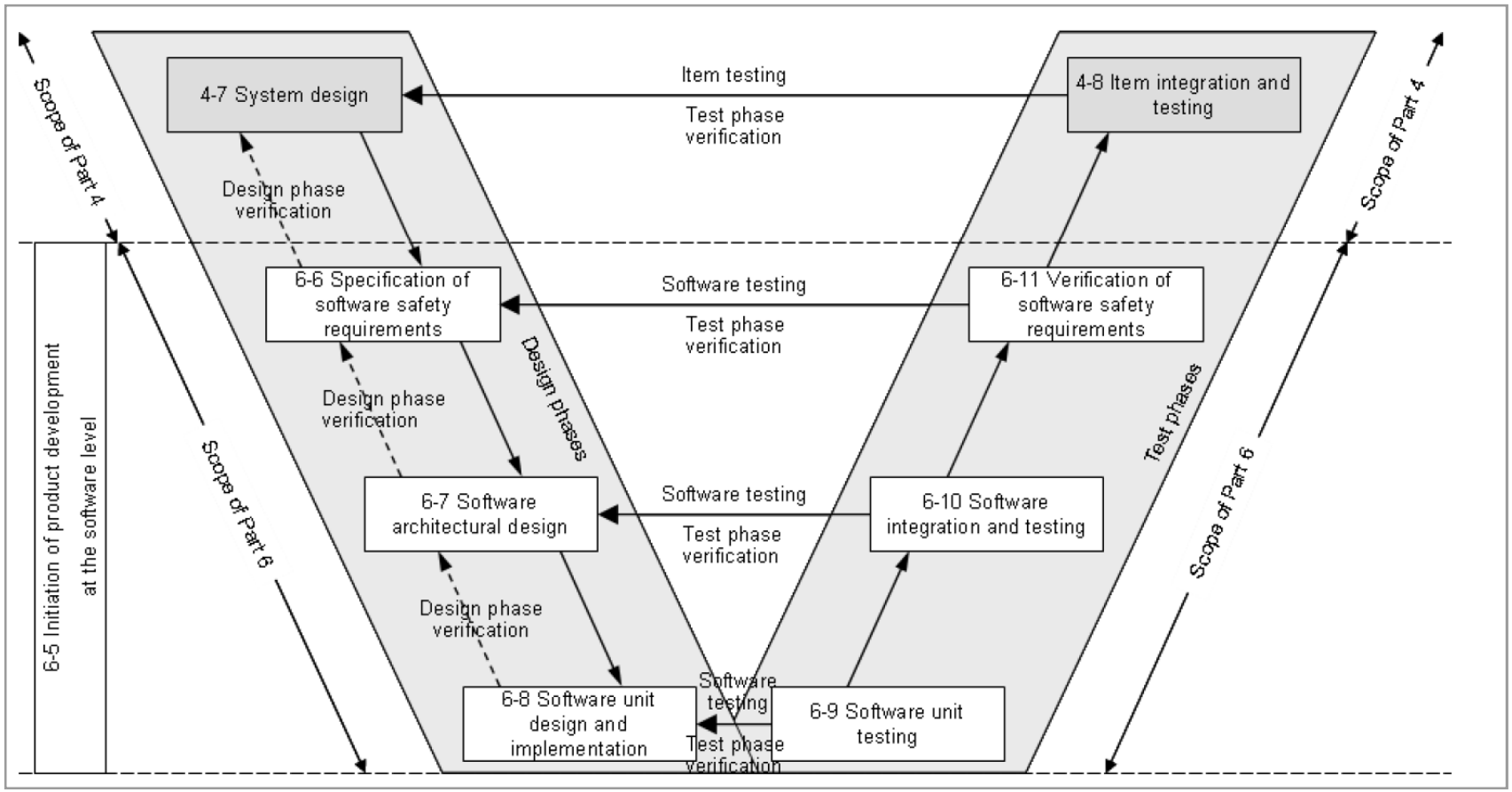
Delivering Software Quality and Security through
Test, Analysis & Requirements Traceability

Part 6 : Product Development – Software Level

- ISO 26262 (Part 6) refers more specifically to the development of software, particularly:
  - Initiation of product development at the software level
  - Derivation of software safety requirements from the system level (following from part 4) and their subsequent verification
  - Software architectural design
  - Software unit design and implementation
  - Software unit testing, and
  - Software integration and testing

6.5 Initiation of Product Development at Software Level

- To plan and initiate the functional safety activities for the sub phases of the software development



6.5 Modelling and Coding Guidelines

- To support the correctness of the design and implementation, the design and coding guidelines for the modelling, or programming languages, shall address following topics:

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

[a] An appropriate compromise of this method with other methods in ISO 26262-6 may be required.

[b] The objectives of method 1b are

- Exclusion of ambiguously defined language constructs which might be interpreted differently by different modellers, programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.
- Exclusion of language constructs which might result in unhandled run-time errors.

[c] The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.

6.6 Specification of Software Safety Requirements

- To specify the software safety requirements which are derived from technical safety concept and system design to detail hardware-software requirements and verify that the software safety requirements are consistent with the technical safety concept and the system design specification

- The technical safety requirements are divided into hardware and software safety requirements. The specification of the software safety requirements considers constraints of the hardware and the impact of these constraints on the software.

6.6 Verification of Software Safety Requirements

- A verification activity shall be performed to demonstrate that the software safety requirements are traceable with the technical safety requirements, the system design and consistent with the relevant parts of the hardware safety requirements achieving complete traceability.

Methods		ASIL			
		A	B	C	D
1a	Informal verification by walkthrough	++	+	o	o
1b	Informal verification by inspection	+	++	++	++
1c	Semi-formal verification[a]	+	+	++	++
1d	Formal verification	o	+	+	+
[a] Method 1c can be supported by executable models.					

6.7 Software Design

- To develop a software architectural design that realizes the software safety requirements and verify the software architectural design achieving bi-directional traceability
- The software architectural design represents all software components and their interactions with one another in a hierarchical structure.

6.7 Software Design

- The software architectural design shall exhibit
  - Modularity
  - Encapsulation
  - Minimum complexity

Methods		ASIL			
		A	B	C	D
1a	Hierarchical structure of software components	++	++	++	++
1b	Restricted size of software components[a]	++	++	++	++
1c	Restricted size of interfaces[a]	+	+	+	+
1d	High cohesion within each software component[b]	+	++	++	++
1e	Restricted coupling between software components[a, b, c]	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts[a, d]	+	+	+	++

[a] In methods 1b, 1c, 1e and 1g "restricted" means to minimise in balance with other design considerations.

[b] Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.

[c] Method 1e addresses the limitation of the external coupling of software components.

[d] Any interrupts used have to be priority-based.

6.7 Verification of Architectural Design

- The software architectural design shall be verified by using the following software architectural design verification methods:

Methods		ASIL			
		A	B	C	D
1a	Informal verification by walkthrough of the design[a]	++	+	o	o
1b	Informal verification by inspection of the design[a]	+	++	++	++
1c	Semi-formal verification by simulating dynamic parts of the design[b]	+	+	+	+
1d	Semi-formal verification by prototype generation / animation	o	o	+	+
1e	Formal verification	o	o	+	+
1f	Control flow analysis[c, d]	+	+	++	++
1g	Data flow analysis[c, d]	+	+	++	++

[a] Informal verification is used to assess whether the software requirements are completely and correctly refined and realised in the software architectural design. In the case of model-based development this method can be applied to the model.

[b] Method 1c requires the usage of executable models for the dynamic parts of the software architecture.

[c] Control and data flow analysis can be carried out informally, semi-formally or formally.

[d] Control and data flow analysis may be limited to safety-related components and their interfaces.

6.8 Software Unit design & implementation

- To specify the software units in accordance with the SW architectural design and the associated SW safety requirements, to implement the software units as specified and to verify the design of the SW units and implementation.

- The specification of the software units shall describe the functional behavior and the internal design.

- The design and implementation of software unit shall achieve
  - Avoidance of unnecessary complexity
  - Testability
  - Maintainability

6.8 Software Unit Design

- The design principles for software unit design and implementation shall be applied to follow below properties

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a, b]	+	++	++	++
1c	Initialisation of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[a, c]	+	++	++	++
1h	No hidden data flow or control flow[b, d]	+	++	++	++
1i	No unconditional jumps[a, c, d]	++	++	++	++
1j	No recursions	+	+	++	++

[a] Methods 1a, 1b, 1c, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

[b] If these compiler features are "tool qualified" in accordance with ISO 26262-8:—, Clause 10, Method 1b need not be applied if a compiler is used which ensures that there will be enough program storage allocated for all dynamic variables and objects before run-time or which inserts run-time tests for correct online-allocation of program storage, i.e. stack bounds checking.

[c] Methods 1g and 1i are not applicable in assembler programming.

[d] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

- Correct execution order
- Interface consistency
- Correct data/control flow
- Simplicity
- Readability and comprehensibility
- Robustness
- Change suitability
- Testability

6.8 Verification of software unit design and implementation

- The software unit design and implementation shall be verified to demonstrate

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Informal verification	See Table 11			
1b	Semi-formal verification[a]	+	+	++	++
1c	Formal verification	o	o	+	+
1d	Control flow analysis[b, c]	+	+	++	++
1e	Data flow analysis[b, c,]	+	+	++	++
1f	Static code analysis	+	++	++	++
1g	Semantic code analysis[d]	+	+	+	+

[a] Method 1b requires an executable design or implementation model of the unit to be verified.

[b] Methods 1d and 1e should be applied at the source code level. These methods are applicable both to manual code development and to model-based development.

[c] Methods 1d and 1e can be part of methods 1c or 1g.

[d] Method 1g is used for mathematical analysis of source code by use of an abstract representation of possible values for the variables. For this it is not necessary to translate and execute the source code.

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Inspection of the software unit design	+	++	++	++
1b	Walkthrough of the software unit design	++	+	O	o
1c	Model Inspection[a]	+	++	++	++
1d	Model Walkthrough[a]	++	+	O	o
1e	Inspection of the source code[b]	+	++	++	++
1f	Walkthrough of the source code[b]	++	+	O	o

[a] In the case of model-based software development the software unit specification can be verified at the model level by applying Method 1c and 1d.

[b] In the case of model-based development with automatic code generation, methods for informal verification of the generated code can be replaced by automated methods and techniques if applicable.

- Compliance with the hardware-software interface
- Completeness regarding the software safety requirements and the software architecture through traceability
- Compliance of the source code with its specification
- Compliance of the source code with the coding guidelines
- Compatibility of the software unit implementations with target hardware.

6.9 Software Unit Testing

- Software units fulfill the software unit specifications and do not contain undesired functionality.

- The following testing methods can be used for proving compliance with specification and Hardware Software interface, correct implementation, absence of unintended functionality, robustness, and sufficiency of the resources.

Methods		ASIL			
		A	B	C	D
1a	Requirement-based test	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[a]	+	+	+	++
1d	Resource usage test[b]	+	+	+	++
1e	Back-to-back test between model and code, if applicable[c]	+	+	++	++

[a] This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting values of variables)

[b] Some aspects of the resource usage test can only by evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

[c] This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

6.9 Test case specification and structural coverage metrics

- To evaluate the completeness of test cases and to demonstrate that there is no unintended functionality, the coverage of requirements at the software unit level shall be determined and the structural coverage shall be measured in accordance with the metrics listed.

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes	+	++	++	++
1c	Analysis of boundary values[a]	+	++	++	++
1d	Error guessing[b]	+	+	+	+

[a] This method applies to interfaces, values approaching and crossing the boundaries and out of range values.

[b] "Error guessing tests" can be based on data collected through a "lessons learned" process and expert judgment.

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Function coverage[a]	+	+	++	++
1b	Call coverage[b]	+	+	++	++

[a] The degree of coverage claimed in method 1a requires at least one execution of every software function. This evidence can be achieved by an appropriate software integration strategy.

[b] The degree of coverage claimed in method 1b requires at least one execution of every software function call.

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

NOTE 1 The structural coverage can be determined by the use of appropriate software tools.

NOTE 2 In the case of model-based development, software unit testing may be moved to the model level using analogous structural coverage metrics for models.

NOTE 3 If instrumented code is used to determine the degree of coverage, it might be necessary to show that the instrumentation has no effect on the test results. This can be done by repeating the tests with non-instrumented code.

NOTE 4 A rationale is to be given for the level of coverage achieved (e.g. for accepted dead code or code segments depending on different software configurations), or else code not covered can be verified using complementary methods (e.g. inspections).

6.10 SW integration & Testing

- To integrate the software components and demonstrate that the software architectural design is correctly realised by the embedded software.
- Integrated software tested against architectural design and interfaces between the software units and the software component.
- Software integration test shall demonstrate
  - Compliance with the software architectural design
  - Compliance with the specification of the hardware-software interface
  - Correct implementation of the functionality
  - Robustness
  - Sufficiency of the resources to support the functionality.

6.11 Verification of SW safety requirements

- To demonstrate that the embedded software fulfils the software safety requirements and embedded software satisfies its requirements in the target environment.
- The results of the verification of the software safety requirements shall be evaluated in accordance with:
  - Compliance with the expected results
  - Coverage of the software safety requirements
  - A pass or fail criteria

Methods		ASIL			
		A	B	C	D
1a	Hardware-in-the-loop	+	+	++	++
1b	Electronic control unit network environments[a]	++	++	++	++
1c	Vehicles	++	++	++	++

[a] Examples are "lab-cars", "rest of the bus" simulations or test benches partially or fully integrating the electrical systems of a vehicle.

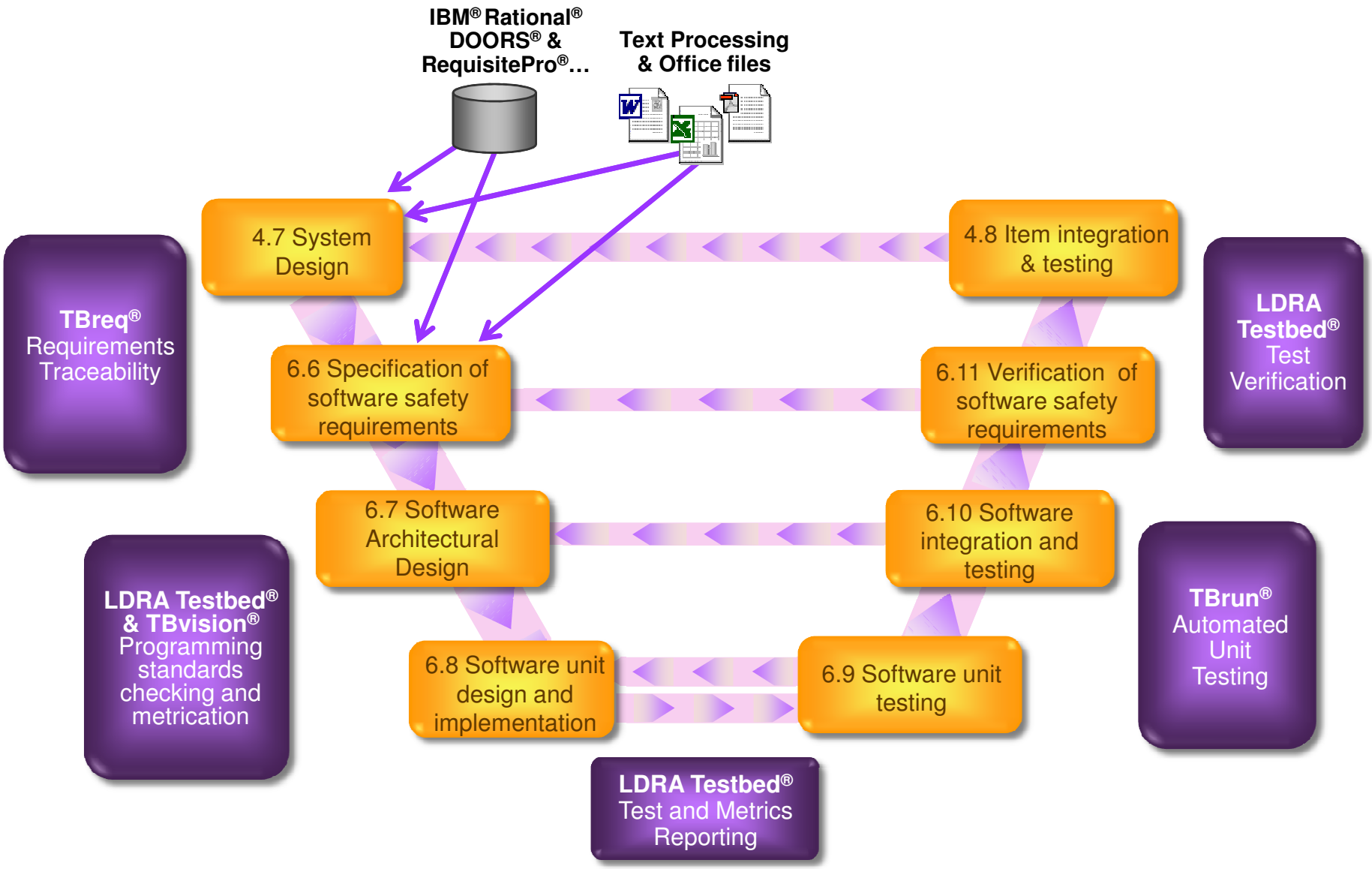
Fitting software tools into ISO 26262 process

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

LDRA Compliance with ISO 26262

- ISO/DIS 26262 is a new standard for the Automotive sector
  - But it is based on principles which have been long established elsewhere
- The concept of adopting Aerospace development principles sounds expensive
  - But tools to handle these issues are
    - Sophisticated and proven, and
    - Designed to apply these principles cost effectively
- Similarly, the LDRA tool suite have long been established to provide the support for IEC 61508 based principles

V Model Process



Static Analysis

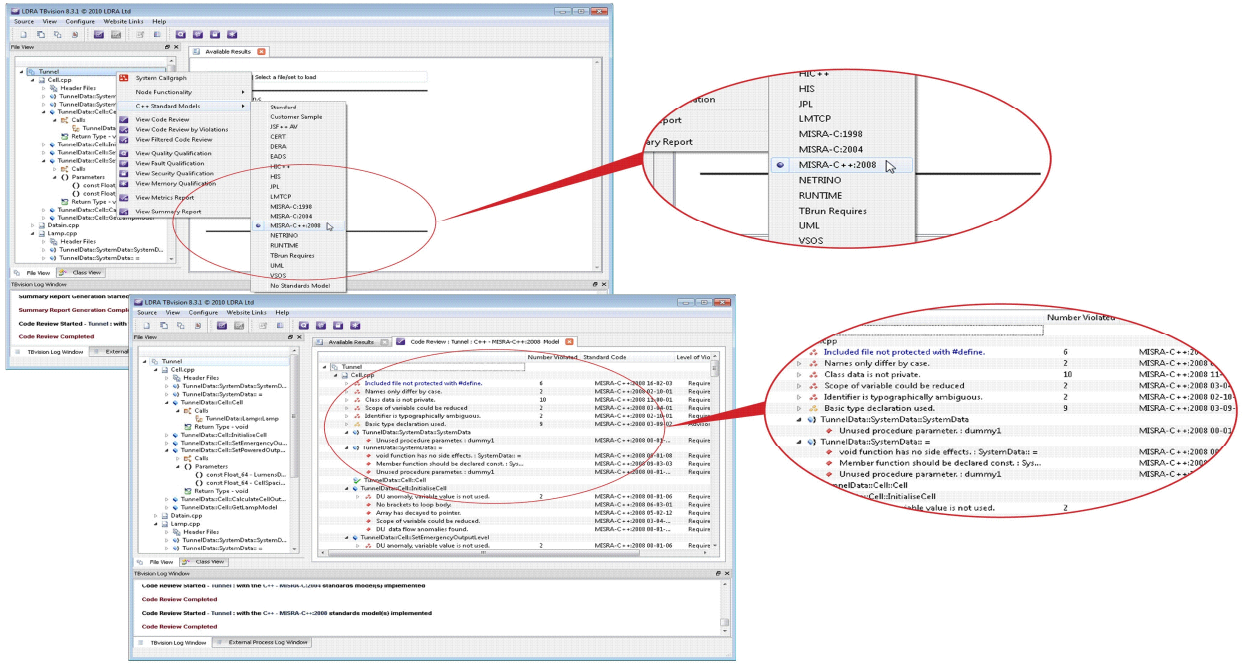
Topics		ASIL			
		A	B	C	D
1a	Informal verification	See table 11 [Figure 7]			
1b	Semi-formal verification	+✔	+✔	++✔	++✔
1c	Formal verification	O	O	+✔	+✔
1d	Control flow analysis	+✔	+✔	++✔	++✔
1e	Data flow analysis,	+✔	+✔	++✔	++✔
1f	Static code analysis	+✔	++✔	++✔	++✔
1g	Semantic code analysis	+✔	+✔	+✔	+✔

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++ ✔	++✔	++✔	++✔
1b	Use of language subsets	++✔	++✔	++✔	++✔
1c	Enforcement of strong typing	++✔	++✔	++✔	++✔
1d	Use of defensive implementation techniques	O	+✔	++✔	++✔
1e	Use of established design principles	+✔	+✔	+✔	++✔
1f	Use of unambiguous graphical representation	+✔	++✔	++✔	++✔
1g	Use of style guides	+✔	++✔	++✔	++✔
1h	Use of naming conventions	++✔	++✔	++✔	++✔

Topics		ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions	++ ✔	++✔	++✔	++✔
1b	No dynamic objects or variables, or else online test during their creation	+✔	++✔	++✔	++✔
1c	Initialisation of variables	++✔	++✔	++✔	++✔
1d	No multiple use of variable names	+✔	+✔	++✔	++✔
1e	Avoid global variables or else justify their usage	+✔	+✔	++✔	++✔
1f	Limited use of pointers	O	+✔	+✔	++✔
1g	No implicit type conversions	+✔	++✔	++✔	++✔
1h	No hidden data flow or control flow	+✔	++✔	++✔	++✔
1i	No unconditional jumps	++✔	++✔	++✔	++✔
1j	No recursions	+✔	+✔	++✔	++✔

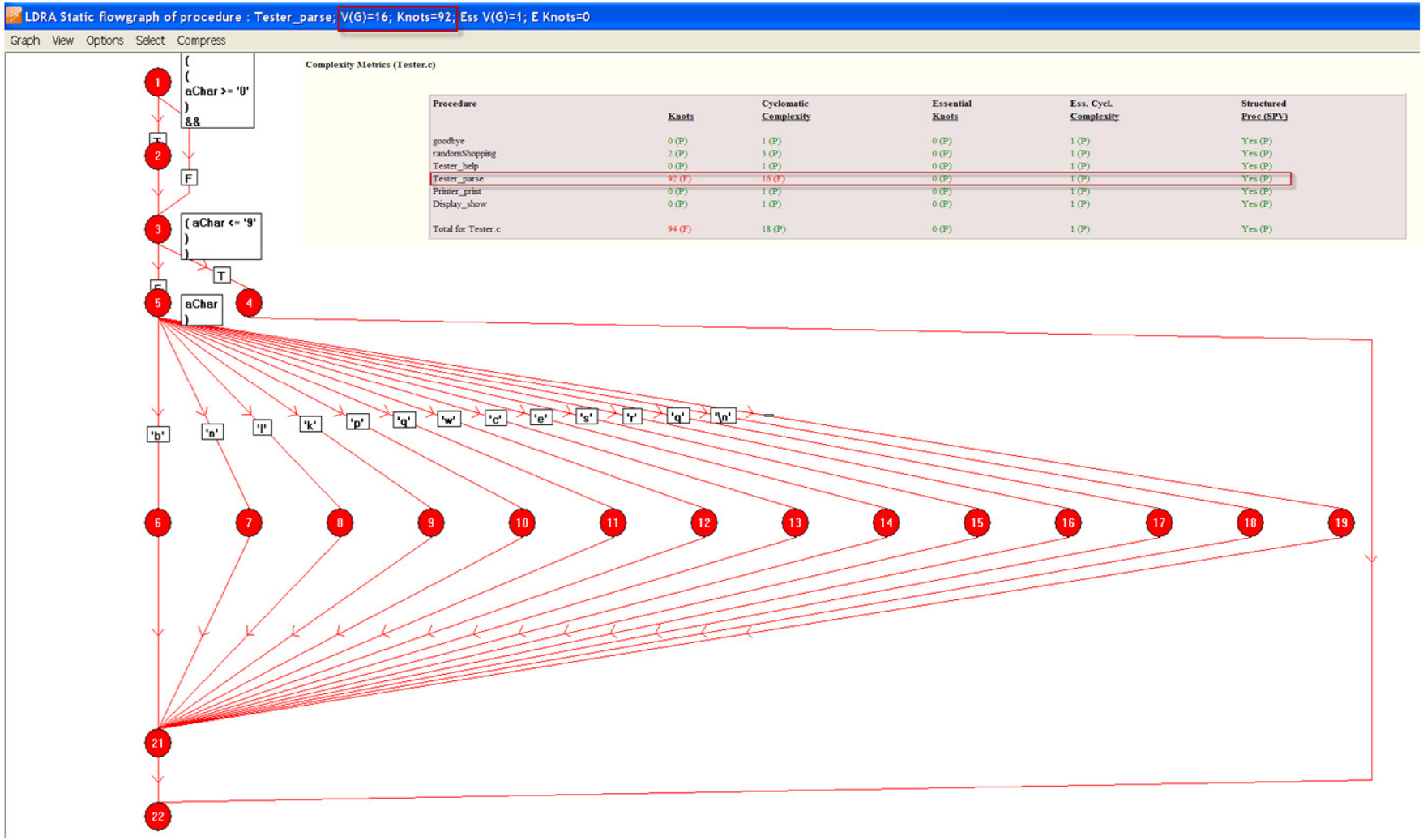
✔ Satisfied by the LDRA tool suite

Coding Standard Enforcement



- JPL
- MISRA
- MISRA-C:2004
- NETRINO
- RUNTIME
- CERT
- CMSE
- CONFORM
- CWE
- DERA
- EADS
- JSF++ AV
- MISRA C++:2008

Cyclomatic Complexity



Software Architectural Design

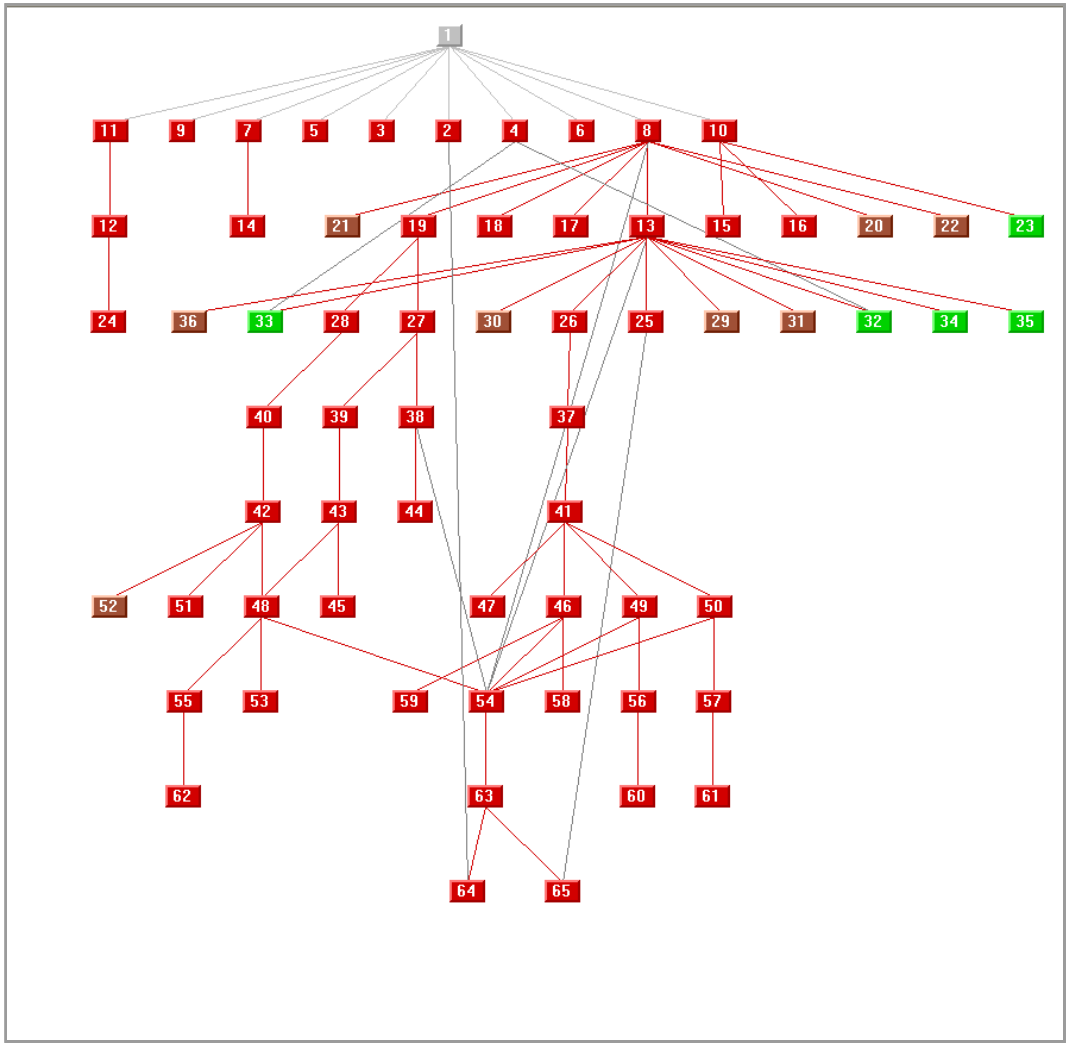
Topics		ASIL			
		A	B	C	D
1a	Hierarchical structure of software components	++✔	++✔	++✔	++✔
1b	Restricted size of software	++✔	++✔	++✔	++✔
1c	Restricted size of interfaces	+✔	+✔	+✔	+✔
1d	High cohesion within each software component	+✔	++✔	++✔	++✔
1e	Restricted coupling between software components,	+✔	++✔	++✔	++✔
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts	+✔	+✔	+✔	++✔

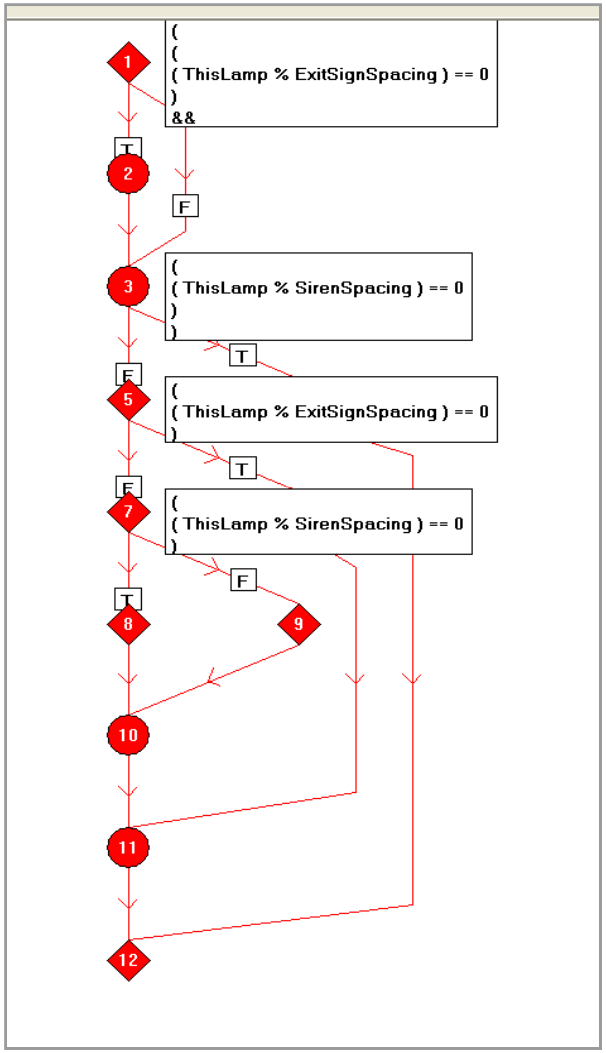
Topics		ASIL			
		A	B	C	D
1a	Plausibility check	++	++	++	++
1b	Detection of data errors	+✔	+✔	+✔	+✔
1c	External monitoring facility	o	+✔	+✔	++✔
1d	Control flow monitoring	o	+✔	++✔	++✔
1e	Diverse software design	o	o	+✔	++✔

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✔ Supported by the LDRA tool suite

Topics		ASIL			
		A	B	C	D
1a	Informal verification by walkthrough of the design	++✔	+✔	o	o
1b	Informal verification by inspection of the design	+✔	++✔	++✔	++✔
1c	Semi-formal verification by simulating dynamic parts of the design	+	+	+	+
1d	Semi-formal verification by prototype generation / animation	o	o	+	+
1e	Formal verification	o	o	+✔	+✔
1f	Control flow analysis	+✔	+✔	++✔	++✔
1g	Data flow analysis	+✔	+✔	++✔	++✔

Data Flow and Control Flow





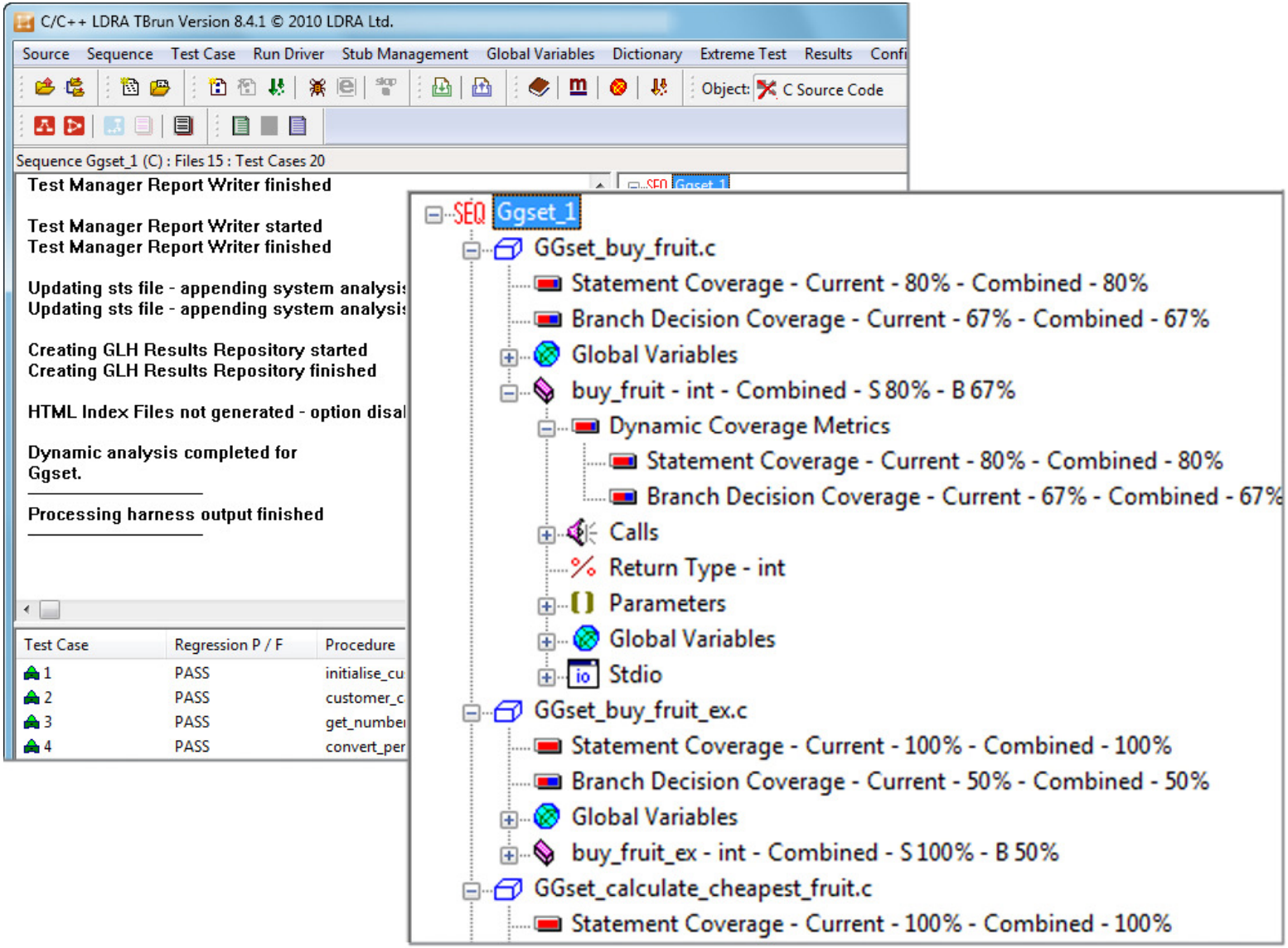
Software Unit Testing

Topics		ASIL			
		A	B	C	D
1a	Requirement-based test	++✔	++✔	++✔	++✔
1b	Interface test	++✔	++✔	++✔	++✔
1c	Fault injection test	+✔	+✔	+✔	++✔
1d	Resource usage test	+	+	+	++
1e	Back-to-back test between model and code, if applicable	+	+	++	++

Topics		ASIL			
		A	B	C	D
1a	Analysis of requirements	++✔	++✔	++✔	++✔
1b	Generation and analysis of equivalence classes	+✔	++✔	++✔	++✔
1c	Analysis of boundary values	+✔	++✔	++✔	++✔
1d	Error guessing	+✔	+✔	+✔	+✔

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✔Satisfied by the LDRA tool suite

Unit Test Case Execution



Coverage Analysis

Topics		ASIL			
		A	B	C	D
1a	Statement coverage	++✔	++✔	+✔	+✔
1b	Branch coverage	+✔	++✔	++✔	++✔
1c	MC/DC (Modified Condition/Decision Coverage)	+✔	+✔	+✔	++✔

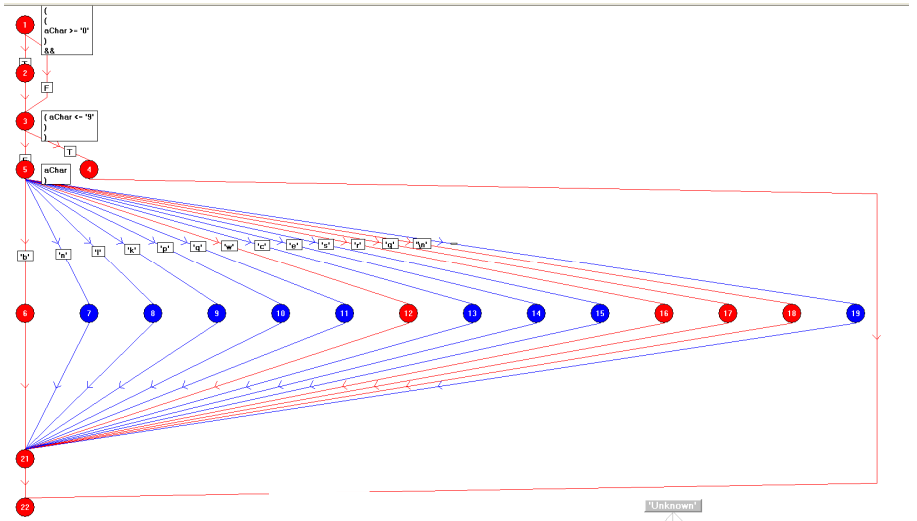
”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✔Satisfied by the LDRA tool suite

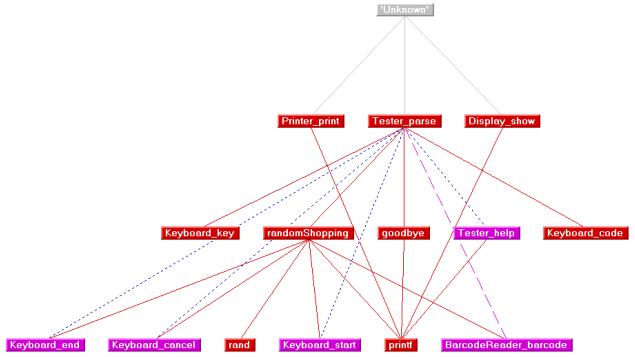
Topics		ASIL			
		A	B	C	D
1a	Function coverage	+✔	+✔	++✔	++✔
1b	Call coverage	+✔	+✔	++✔	++✔

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✔Satisfied by the LDRA tool suite

Coverage Analysis

```
328 (98)            if                                                  0       14      14
329                   (                                                 0       14      14
330                   (                                                 0       14      14
331                   aChar >= '0'                                      0       14      14
332                   )                                                 0       14      14
333                   &&                                                0       14      14
334                   ( aChar <= '9'                                    0       10      10
335                   )                                                 0       10      10
336                   )                                                 0       14      14
337 (99)              {                                                 0       5       5
338 (107)           /* LDRA_INSPECTED 203 S : cast is ok                -       -       -
339 (108)       */
340 (101)             Keyboard_key (                                    0       5       5
341                   ( LDRA_uint32_t ) aChar - 48U ) ;                 0       5       5
342 (102)           }                                                   0       5       5
343 (103)           else                                                0       5       5
344 (104)             {                                                 0       9       9
345 (105)               switch (                                        0       9       9
346                       aChar                                         0       9       9
347                       )                                             0       9       9
348 (106)                 {                                           0       9       9
349 (107)                 case 'b' :                                  0       1       1
350 (108)                   Keyboard_code () ;                        0       1       1
351 (109)                   break ;                                   0       1       1
352 (110)                 case 'n' :                                  0       0  ***  0  ***
353 (111)                   BarcodeReader_barcode ( 12345U ) ;        0       0  ***  0  ***
354 (112)                   break ;                                   0       0  ***  0  ***
355 (113)                 case 'l' :                                  0       0  ***  0  ***
356 (114)                   BarcodeReader_barcode ( 12346U ) ;        0       0  ***  0  ***
357 (115)                   break ;                                   0       0  ***  0  ***
358 (116)                 case 'k' :                                  0       0  ***  0  ***
359 (117)                   BarcodeReader_barcode ( 12347U ) ;        0       0  ***  0  ***
360 (118)                   break ;                                   0       0  ***  0  ***
361 (119)                 case 'p' :                                  0       0  ***  0  ***
362 (120)                   BarcodeReader_barcode ( 12348U ) ;        0       0  ***  0  ***
363 (121)                   break ;                                   0       0  ***  0  ***
364 (122)                 case 'q' :                                  0       0  ***  0  ***
365 (123)                   BarcodeReader_barcode ( 12349U ) ;        0       0  ***  0  ***
366 (124)                   break ;                                   0       0  ***  0  ***
367 (125)                 case 'w' :                                  0       1       1
368 (126)                   BarcodeReader_barcode ( 12350U ) ;        0       1       1
369 (127)                   break ;                                   0       1       1
370 (128)                 case 'c' :                                  0       0  ***  0  ***
371 (129)                   Keyboard_cancel () ;                      0       0  ***  0  ***
372 (130)                   break ;                                   0       0  ***  0  ***
373 (131)                 case 'e' :                                  0       0  ***  0  ***
374 (132)                   Keyboard_end () ;                         0       0  ***  0  ***
375 (133)                   break ;                                   0       0  ***  0  ***
376 (134)                 case 's' :                                  0       0  ***  0  ***
377 (135)                   Keyboard_start () ;                       0       0  ***  0  ***
378 (136)                   break ;                                   0       0  ***  0  ***
379 (137)                 case 'r' :                                  0       2       2
```



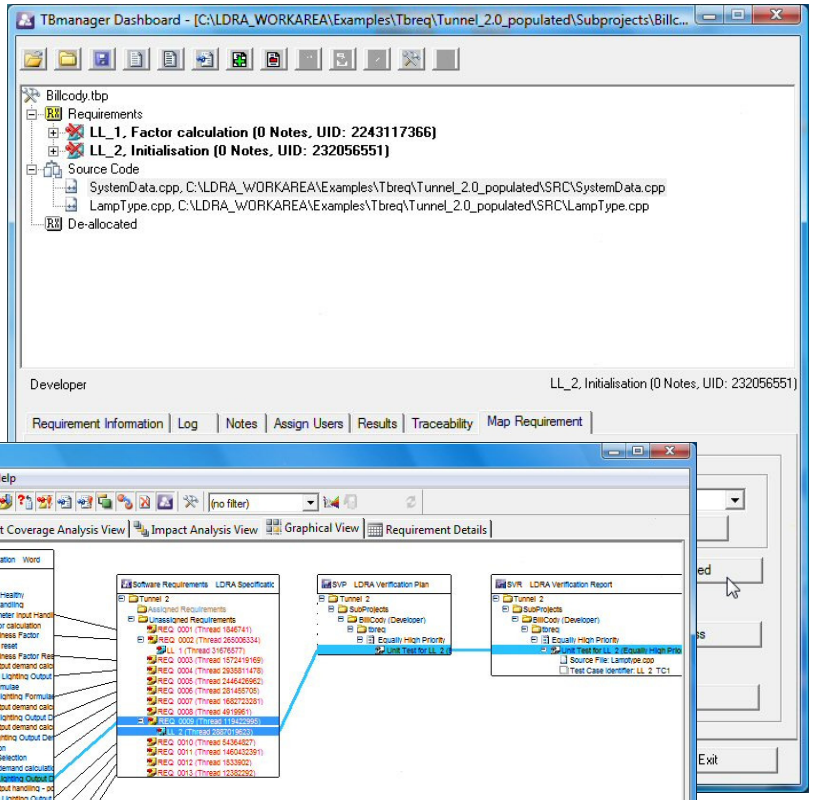


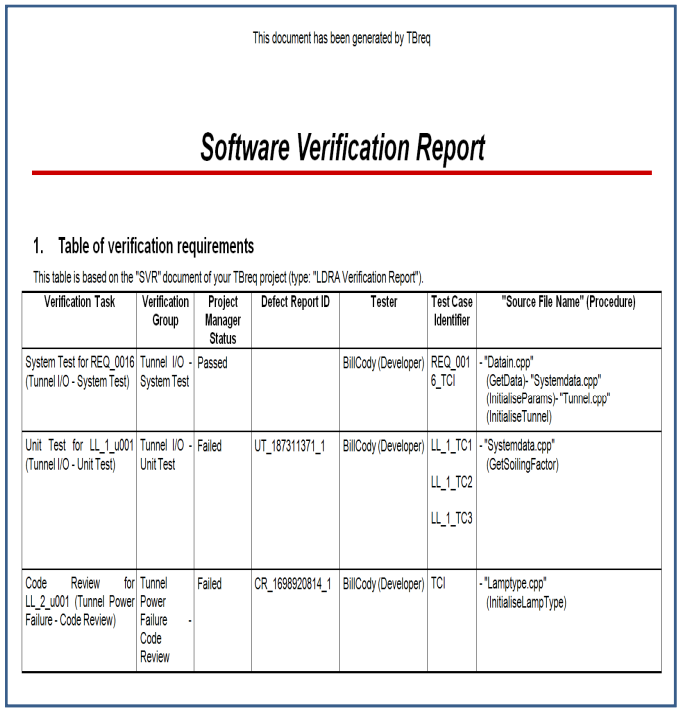
Statement (TER1) = 76 % (Fail) Branch/Decision (TER2) = 52 % (Fail) LCSAJ (TER3) = 52 % (Fail)
MC/DC = 100 % (Pass) Call/Return = 78 % (Pass)

Procedure	Statement	Branch	LCSAJ	MC/DC
goodbye	+100	[No Branches]	+100	[No BCs]
randomShopping	+100	+100	+78	[No BCs]
Tester_help	+100	[No Branches]	0	[No BCs]
Tester_parse	+58	+44	+44	+100
Printer_print	+100	[No Branches]	+100	[No BCs]
Display_show	+100	[No Branches]	+100	[No BCs]

Traceability Matrix

- Forward and Backward Traceability





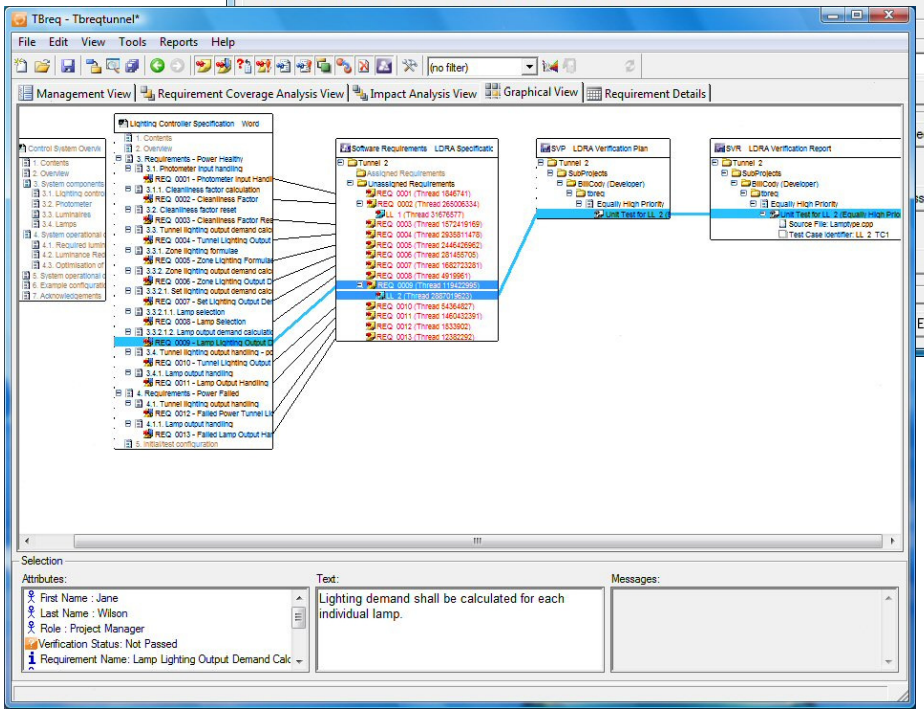
This document has been generated by TBreq

Software Verification Report

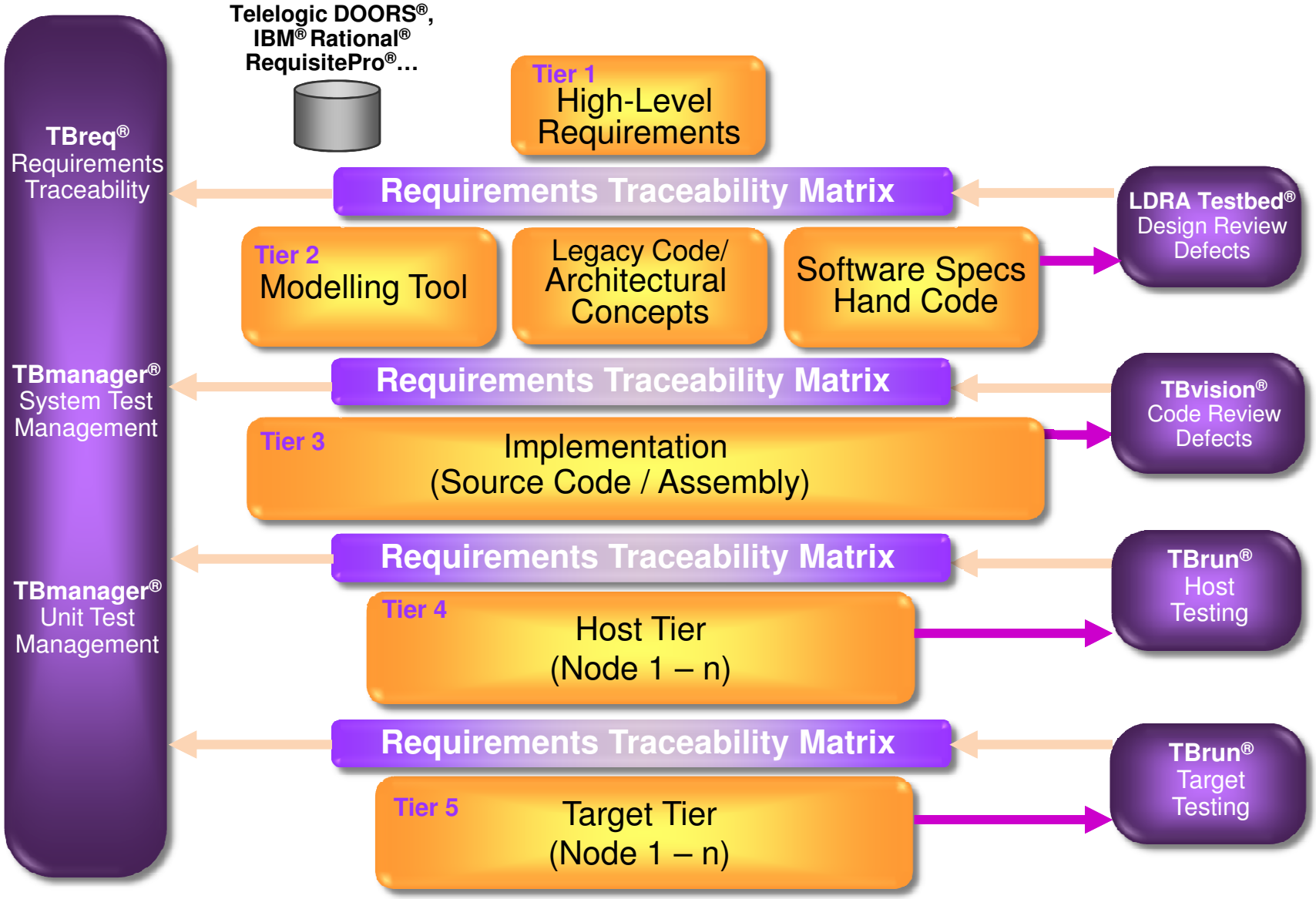
1. Table of verification requirements

This table is based on the "SVR" document of your TBreq project (type: "LDRA Verification Report").

Verification Task	Verification Group	Project Manager Status	Defect Report ID	Tester	Test Case Identifier	"Source File Name" (Procedure)
System Test for REQ_0016 (Tunnel I/O - System Test)	Tunnel I/O - System Test	Passed		BillCody (Developer)	REQ_0016_TC1	- "Datain.cpp" (GetData)- "Systemdata.cpp" (InitialiseParams)- "Tunnel.cpp" (InitialiseTunnel)
Unit Test for LL_1_u001 (Tunnel I/O - Unit Test)	Tunnel I/O - Unit Test	Failed	UT_187311371_1	BillCody (Developer)	LL_1_TC1 LL_1_TC2 LL_1_TC3	- "Systemdata.cpp" (GetSoilingFactor)
Code Review for LL_2_u001 (Tunnel Power Failure - Code Review)	Tunnel Power Failure - Code Review	Failed	CR_1698920814_1	BillCody (Developer)	TC1	- "Lamptype.cpp" (InitialiseLampType)



Software Lifecycle Traceability



Conclusion

- IEC 26262 ensures the safety of the electrical / electronic / programmable electronic devices in Road vehicles.
- Using tools with a proven track record and pedigree to automate the software development process:
  - Will help in producing safe product
  - Provides confidence to the manufacturers
  - Save time and money

For further information visit:

www.ldra.com

india@ldra.com

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