Implementing the EN 5012X standards with the LDRA tool suite®

Cost effective software certification from Basic Integrity to SIL 4

www.ldra.com

* Registration required to download the document

Contents

Background

The approval process of interlocking systems mandates adherence to the CENELEC standards EN 50126-1[1], EN 50126-2[2], EN 50128[3] and EN 50129[4] across European countries. Collectively they describe the life cycle process for safety relevant Guided Transport Systems (GTS).

Like the automotive, medical device and process industries, the railway sector based their functional standard on the industry agnostic functional safety standard IEC 61508[5]. The resulting EN 5012X series has become the dominant railway functional safety standard, and its requirements and processes are becoming increasingly familiar across the world. The international standards IEC 62278[6], IEC 62779[7], and IEC 62280[8] very largely mirror EN 50126, EN 50128 and EN 50129 respectively and can be considered to be identical in the context of this document.

Many of the practices adopted by IEC 61508 and its derivatives can be traced to the commercial and defence avionics industries. The RTCA Inc. series standards such as DO-178B/C[9], DO-278A[10] and their supplements have proven that adherence to a structured set of best practices results in large scale reliable systems that protect public safety. Although EN 5012X even in its original form was a relatively recent innovation, this history is significant because the establishment of functional safety standards elsewhere gave rise to a sophisticated industry providing support for their effective application. Consequently, the GTS industry benefits from established and proven tools and techniques predating EN 5012X itself.

This document describes the key software development and verification process activities of the standards, and uses LDRA's tool suite to show how automation can assist in proving compliance in a cost-effective manner. Extracts from the EN 5012X series are shown in *italics*.

[1] EN 50126-1:2017 Railway Applications. The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS). Generic RAMS Process
[2] EN 50126-2:2017 Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 2: Systems Approach to Safety
[3] EN 50128:2011+A2:2020 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems
[4] EN 50129:2017 Railway applications. Communication, signalling and processing systems. Safety related electronic systems for signalling
[5] IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems
[6] IEC 62278:2002 Railway applications - Specification and demonstration of reliability, availability, maintainability and safety (RAMS)
[7] IEC 62279:2015 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems
[8] IEC 62280:2014 Railway applications - Communication, signalling and processing systems - Safety related communication in transmission systems
[9] RTCA DO-178C Software Considerations in Airborne Systems and Equipment Certification, Prepared by SC-205, December 13, 2011
[10] RTCA DO-278 Guidelines for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance

Software in the context of the EN 5012X standards

Each of the EN 5012X series of standards has an impact on the software development lifecycle.

EN 50126-1 Railway Applications. The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS). Generic RAMS Process[11]	EN 50126 is relevant to software systems, but not specifically focused upon them. It defines the four terms Reliability, Availability, Maintainability and Safety, and describes their interaction and their management. It also defines a systematic process for specifying requirements for RAMS and demonstrating that those requirements are achieved.
EN 50126-2 Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 2: Systems Approach to Safety	
EN 50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems	EN 50128 focuses specifically on software systems and their environment. It specifies procedures and technical requirements for the development of safety related programmable electronic systems for use in railway control and protection applications.
EN 50129 Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling	EN 50129 is relevant to software systems, but not specifically focused upon them. EN 50129 specifies the lifecycle activities which are to be completed before the acceptance stage, and the activities to be carried out after it. It is primarily concerned with the evidence to be presented for the acceptance of safety-related systems.

Figure 1: Software development in the context of the EN 5012X series of standards

Safety Integrity Levels (SILs)

Like DO-178B/C and IEC 61508 before it, EN 50129 specifies a number of hazard classifications levels – in this case, known as SILs (Safety Integrity Levels). Development process checks and safety measures are specified to avoid an unreasonable residual risk proportionate to the SIL. SILs range from Basic Integrity and then from 1 through to 4, where SIL 4 represents the most hazardous and hence demanding level so that the overhead involved in producing a safety critical SIL 4 system (e.g. braking) is significantly greater than that required to produce a system with fewer safety implications (e.g. interior passenger lighting).

EN 50129 discusses the derivation and assignment of SILs at some length. In brief, a SIL can be assigned to any safety-related system, sub-system or component performing a safety relevant function. The process involved with such an assignment starts with the identification of potential hazardous events. Then a Tolerable Hazard Rate (THR) is assigned for each hazardous event that might occur as a result of malfunction or failure of function, expressed as a probability per unit of time and taking into account risk reduction measures designed to reduce the rate of occurrence.

Each hazard is then associated with a functional failure of a function or set of functions, and the THR used to derive a Tolerable Function Failure Rate (TFFR), from which a SIL can be assigned.

[11] EN 50126-1:2017 Railway Applications. The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS). Generic RAMS Process

TFFR per hour and per function	Safety Integrity Level
$10^{-9} \leq TFFR < 10^{-8}$	4
$10^{-8} \leq TFFR < 10^{-7}$	3
$10^{-7} \leq TFFR < 10^{-6}$	2
$10^{-6} \leq TFFR < 10^{-5}$	1

Figure 2: Table A.1 The SIL table. Source: EN 50129:2018

From the specific perspective of software development, the SIL assigned to each software component has a considerable impact on its verification and validation activity and the overhead associated with it.

Changes resulting from EN 50128:2011/A2 (Amendment 2)

Changes to the standard resulting from Amendment 2 are editorial only, and mostly designed to align the document with EN 50126-1:2017, EN 50126-2:2017 and EN 50129:2018. The most notable change is that “SIL 0” is replaced by the term “Basic Integrity”, where the Basic Integrity requirements are to be fulfilled for the software part of functions that have a safety impact below SIL 1.

Security in the context of the EN 5012X standards

Most embedded software used in the GTS sector has not taken security requirements into account, simply because security and connectivity have not really been on the agenda. Embedded applications have tended to be static, fixed function, device specific implementations. Isolation has been a sufficient guarantee of security for many years, and practices and processes have relied on that status.

Perhaps because of this traditional isolation of railway systems, EN 5012X makes no specific mention of cybersecurity; indeed, in the field of safety critical embedded software, security concerns have generally been perceived to be a separate domain from the core business of functional safety. Yet if hackers have the potential to remotely control (for example) braking and signalling systems, then security vulnerabilities clearly put safety at risk. In this situation safety and security are indistinguishable.

As soon as there is potential for security vulnerabilities to threaten safety, EN 50128 demands that they are dealt with as for any other hazards. It demands that SILs are appropriately assigned, that associated software is designed with due reference to those classifications, and that it is developed and verified to show compliance with the system’s safety requirements. In short, the action to be taken to deal with each safety-threatening security issue needs to be proportionate to the risk (and hence SIL).

In acknowledgement of that, IEC 62334-4-1 is gaining increasing prominence in an industry looking for more focused cybersecurity related guidance. IEC 62443 is a series of multi-industry standards defining cybersecurity protection methods and techniques categorised to apply to all stakeholders including manufacturers, asset owners and suppliers. The fourth in the series, IEC 62443-4:2018, specifies the requirements for the secure development of systems used in industrial control and automation. It defines secure development life-cycle requirements related to cybersecurity for

products intended for use in Industrial Automation and Control Systems (IACS) which are sufficiently complementary to those of EN 5012X to be integrated into the same software development lifecycle.

The IEC 62443-4-1 development life-cycle phases include security requirements definition, secure design, secure implementation (including coding guidelines), verification and validation, defect management, patch management and product end-of-life. These activities and tasks can be applied to new or existing processes for developing, maintaining, and retiring hardware, software, or firmware.

EN 5012X process objectives

EN 50128 §5.3.2.1 requires that *"a lifecycle model for the development of software shall be selected"* but stops short of dictating which model that should be, requiring only that it is to be detailed in the Software Quality Assurance Plan. It does include examples of appropriate models, one of which is a V-model (Figure 3).

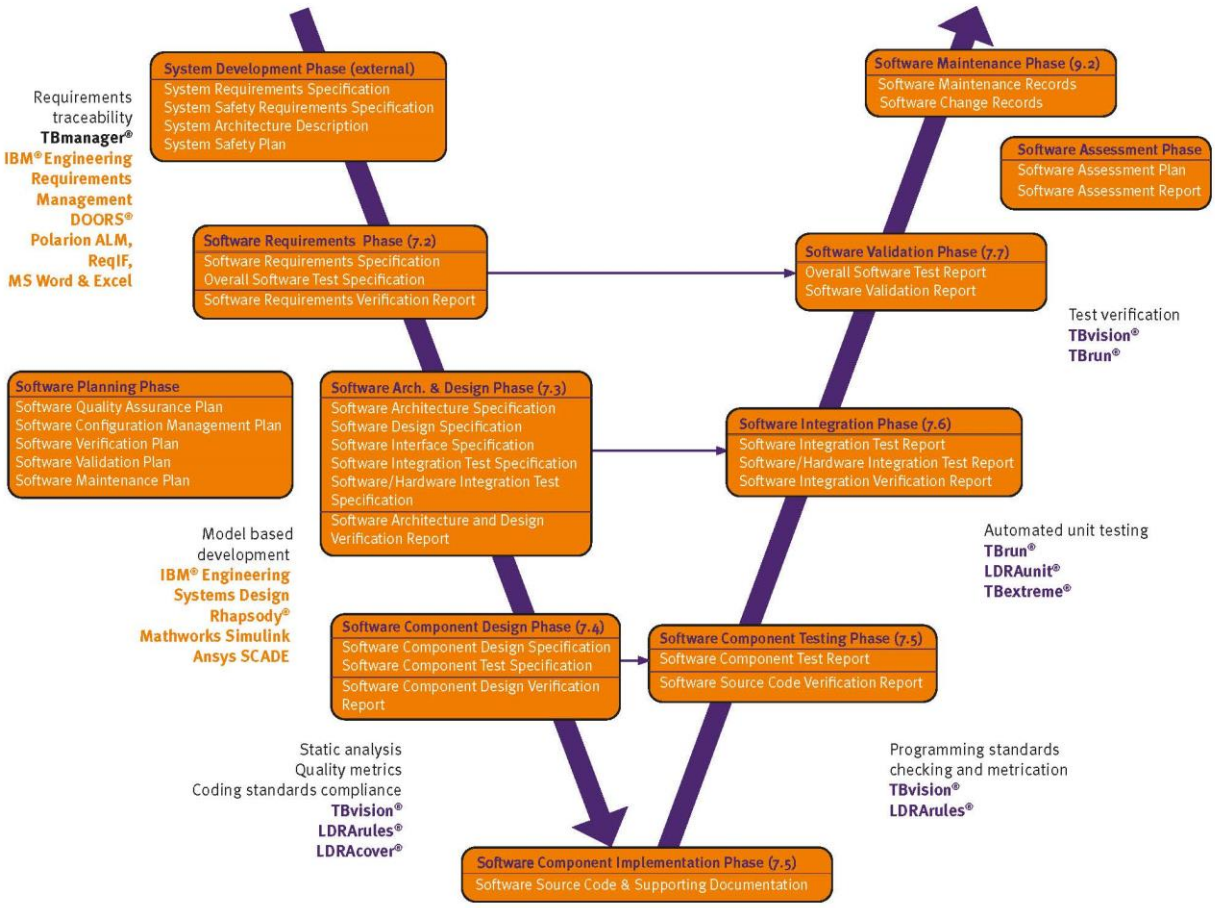

Figure 3: Mapping the capabilities of an automated tool chain to the V-model example of an appropriate EN 50128 lifecycle model.

These different elements are not intended to be viewed as isolated silos; indeed, EN 50126 requires the safety team to identify the safety-related functional requirements and the safety-integrity requirements, and that they follow all of the requirements throughout the whole of the cycle – so that requirements are traceable to architectural design, that architectural design is traceable to component design and implementation, and so on.

Traceability is at the heart of EN 5012X with (for example) Table B.7 of EN 50128 requiring that the nominated Validator shall not only *"review the software requirements against the intended*

environment/use" but also *"review that developed solutions are traceable to the software requirements"*. This bidirectional traceability is designed to reduce the risk of failure by ensuring that not only is all required functionality present and proven, but also that there is no redundant code or "feature creep".

Once software safety requirements and architecture are defined, the software units can be designed and then implemented in accordance with that design. The developing organization is required to establish coding rules appropriate to the circumstances of the project, and the source code for the implemented units must be validated to ensure that those coding rules are adhered to. The software units must then be dynamically tested (executed) to demonstrate that they fulfil the software unit design specification and do not include unspecified functionality.

Structural coverage analysis is then required to determine which code structures and component interfaces have not been exercised during execution of these requirements-based test procedures. Unexecuted portions of code require further analysis resulting in addition or modifications of test cases, changes to inadequate requirements, removal of dead or deactivated code, or unintended functionality.

Variations in low level implementation details such as endianness and the sizes of data and address words can result in differing behaviour between test and target environments. EN 50128 §8.4.5.1 requires that the test environment shall correspond as closely as possible to the target environment.

Although it provides extensive guidelines relating to the use of software tools, EN 5012X does not require that they are used. However, for all but very trivial applications, attempting to meet the standard without some level of automation would be a disproportionately labour intensive task. Tools are available to assist with almost all aspects of EN 50128 not only to automate the tests themselves, but also to provide documentary evidence (or "artefacts") relating to their successful completion.

Automating EN 5012X processes

Software tools effectively leverage the experience and expertise of their vendors in this and other industries to ease the path to certification. Tools can be applied to the EN 5012X processes as illustrated in Figure 3. Each element of the V-model diagram is discussed in the following clauses, and relevant tables from the EN 5012X standard expanded and referenced.

For each phase, EN 50128 identifies several items of documentation to be created, including a verification report at the end of the phase. The verification reports draw on information created both during and prior to the current phase.

Automating traceability between these different artefacts can help to make the process easier to manage and less error prone, especially when changes occur or when tests highlight design problems demanding that earlier phases are to be revisited. The TBmanager component of the LDRA tool suite can be very helpful in this regard.

Software planning phase (EN 50128 §5.3)

The V-model shows planning documentation as offset from the lifecycle itself. In discussing *"Lifecycle issues and documentation",* EN 50128 §5.3.2.4 requires that *"The Software Quality Assurance Plan, Software Verification Plan, Software Validation Plan and Software Configuration Management Plan shall be drawn up at the start of the project and maintained throughout the software development life cycle."* Each is therefore available as input from the software lifecycle phase onwards.

System development phase (EN 50126)

The system development phase as represented in the EN 50128 V-model suggests that four design and test documents are to be produced, as described in the following EN 5012X sections:

EN 50128 §7.2 Specification of system requirements
EN 50126-2 §9 Specification of system safety requirements
EN 50126-1 §7.6 Architecture and apportionment of system requirements
EN 50126-1 §7.3.2.3 Safety plan

Although EN 50128 is the primary document for software development, it needs to be considered in the context of the system design for the product as a whole which is the domain of EN 50126. Clearly the system requirements, system safety requirements, architectural design, and safety plan involve software considerations, but in this early part of the development lifecycle requirements are handled within one development process whether they impact software, hardware, or both in the form of the hardware-software interface.

The products of this design phase potentially include CAD drawings, spreadsheets, textual documents and many other artefacts, and clearly a variety of tools can be involved in their production. The management of the status of each of those elements and maintaining traceability between them and subsequent phases can cause a project management headache, and the principles of traceability are established early with the insistence that "*For each document, traceability shall be provided in terms of a unique reference number (including version) including a defined and documented relationship with other documents"* (EN 20126-1 §6.6).

The ideal tools for requirements management depends largely on the scale of the development. If there are few developers in a local office, a simple spreadsheet or Microsoft Word document may suffice. Bigger projects, perhaps with contributors in geographically diverse locations, are likely to benefit from an Application Lifecycle Management (ALM) tool such as IBM Engineering Requirements Management DOORS[12], Siemens PLM Polarion ALM[13], or other ALM tools supporting standard Requirements Interchange Formats[14].

Software requirements phase (EN 50128 §7.2)

The software requirements phase as represented in the EN 50128 V-model suggests that three design and test documents are to be produced, as described in the following EN 5012X sections:

EN 50128 §7.2 Software requirements specification
EN 50128 §7.2.4.17 to §7.2.4.19 Overall software test specification
EN 50128 §7.2.4.20 to §7.2.4.22 Software requirements verification report

The software requirements verification report is written at the end of the phase on the basis of the other listed documents, plus the system safety requirements and the software quality assurance plan.

The objectives of this phase are "*To describe a complete set of requirements for the software meeting all System and Safety Requirements and provide a comprehensive set of documents for each subsequent phase"* (EN 50128 §7.2.1.1). The specification of the software safety requirements considers constraints of the hardware and the impact of these constraints on the software, but primarily focuses on the specification of software requirements to support the subsequent design phases.

[12] IBM Engineering Requirements Management DOORS Family https://www.ibm.com/uk-en/products/requirements-management
[13] Siemens Polarion https://polarion.plm.automation.siemens.com/
[14] Object Management Group Requirements Interchange Format http://www.omg.org/spec/ReqIF/

This phase is essentially the interface between the product-wide system design of EN 50126, and the software focused EN 50128. It details the process of evolution of lower level requirements as they relate specifically to the software system. That implies a continued leveraging of the requirements management tools selected for the project, as discussed in relation to the system development phase.

Software architecture and design phase (EN 50128 §7.3)

The software architecture and design phase as represented in the EN 50128 V-model suggests that six design and test documents are to be produced, as described in the following EN 5012X sections:

EN 50128 §7.3.4.1 to §7.3.4.14 Software Architecture Specification
EN 50128 §7.3.4.20 to §7.3.4.24 Software Design Specification
EN 50128 §7.3.4.18 to §7.3.4.19 Software Interface Specification
EN 50128 §7.3.4.29 to §7.3.4.32 Software Integration Test Specification
EN 50128 §7.3.4.34 to §7.3.4.39 Software/Hardware Integration Test Specification
EN 50128 §7.3.4.40 to §7.3.4.43 Software Architecture and Design Verification Report

The Software Architecture and Design Verification Report is written at the end of the phase on the basis of the other listed documents, plus the software requirements specification.

The objectives of the software architecture and design phase are concerned with the *"development of a software architecture that achieves the requirements of the software"* (EN 50128 §7.3.1.1) whilst giving due consideration to hardware/software interactions, the selection of an appropriate design method, adherence to software safety integrity levels, and the testability of the resulting system.

There are many tools available for the generation of the software architectural design, with graphical representation of that design an increasingly popular approach. Appropriate tools are exemplified by IBM Engineering Systems Design Rhapsody[15], MathWorks Simulink[16] and Ansys SCADE[17].

EN 50128 §7.3.4.14 requires that "*7.3.4.14 The Software Architecture Specification shall choose techniques and measures from Table A.3*". Table A.3 describes characteristics that are to be part of the architecture of the system which are not to be confused with test processes or activities designed to seek out related anomalies. For example, Row 1 of the table references EN 50128 §D.14 which in turn refers to the production of "*programs which detect anomalous control flow, data flow or data values during their execution and react to them in a predetermined and acceptable manner*". The LDRA tool suite could be used to show that such objectives are being fulfilled using the techniques discussed in this paper.

Static analysis tools also have a part to play in the verification of the design in the form of control and data flow analysis of the code generated in accordance with it. As shown in Figure 4, the tools derive the relationship between some or all of the code components and represent it graphically such that it can then be compared with the intended design.

[15] IBM Engineering Systems Design Rhapsody https://www.ibm.com/uk-en/products/systems-design-rhapsody
[16] Mathworks Simulink Simulation and Model-Based Design https://www.mathworks.com/products/simulink.html
[17] Ansys SCADE Suite https://www.ansys.com/products/embedded-software/ansys-scade-suite

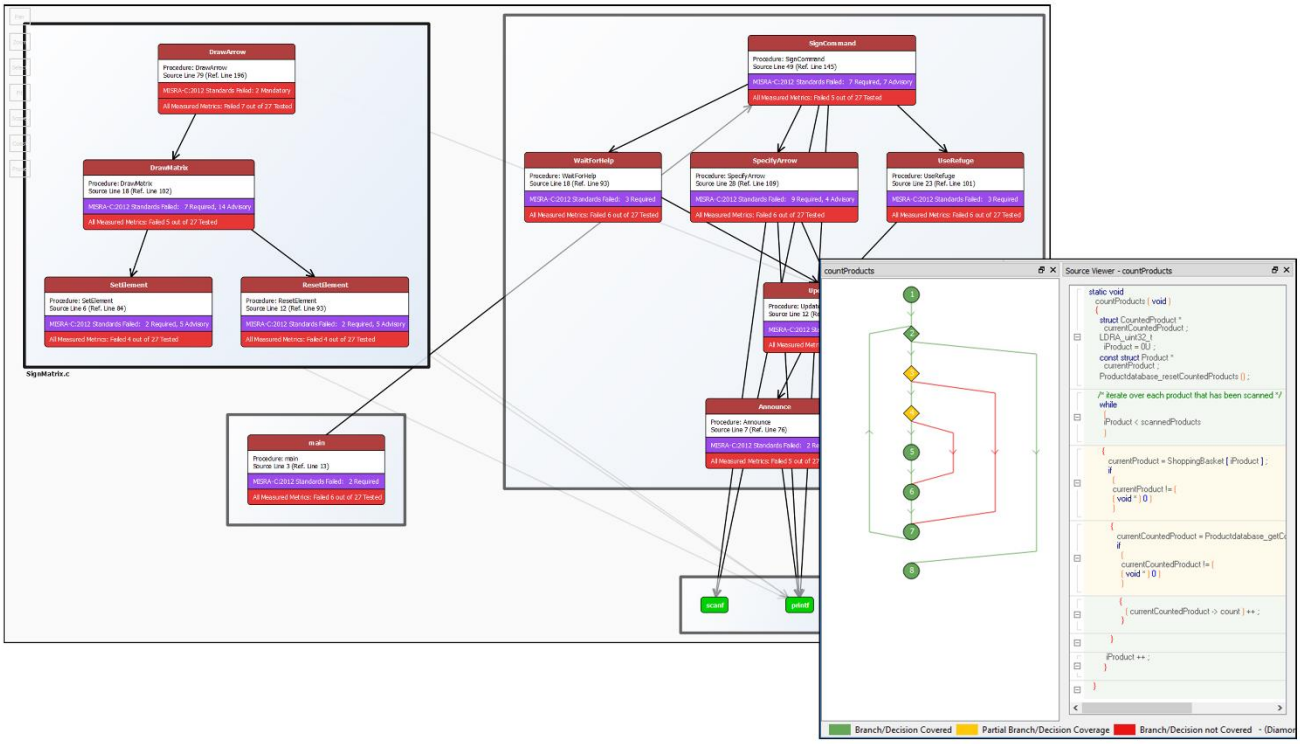

Figure 4: Diagrammatic representations of control and data flow generated from source code by the LDRA tool suite aid verification of software architectural design

Reverse engineering

Much of the flow of the EN 50128 processes is based on an overriding presumption that the project begins with nothing. However, it is entirely possible that there is a need to adapt a long established application predating current standards, or a need to integrate third-party source code into an existing design. What if source code exists, but design documentation doesn't?

In such circumstances graphical representations of the code (Figure 4) can help to establish the architecture of the existing system, such that requirements and designs can be established retrospectively. Updates can then be designed and proven in accordance with EN 50128 principles.

Model based development

The LDRA tool suite can be integrated with several different model based development tools, one such example being MathWorks Simulink. The development phase itself involves the creation of the model in the usual way, with the integration becoming more pertinent once source code has been auto generated from that model.

The integration itself is primarily leveraged during software unit testing, and software integration and testing. The topic of model based development is therefore revisited later in this document in relation to those sub-phases.

Software component design phase (EN 50128 §7.4)

The software component design phase as represented in the EN 50128 V-model suggests that three design and test documents are to be produced, as described in the following EN 5012X sections:

EN 50128 §7.4.4.1 to §7.3.4.6 Software Component Design Specification
EN 50128 §7.4.4.7 to §7.3.4.10 Software Component Test Specification
EN 50128 §7.3.4.11 to §7.3.4.13 Software Component Design Verification Report

The Software Component Design Verification Report is written at the end of the phase on the basis of the other listed documents, plus the software design specification.

The objectives of this phase are "*To develop a software component design that achieves the requirements of the Software Design Specification to the extent required by the software safety integrity level"* (EN 50128 §7.2.1.1) and "*to develop a Software Component Test Specification that*

achieves the requirements of the Software Component Design Specification to the extent required by the software safety integrity level" (EN 50128 §7.2.1.2).

The Software Component Design Specification is required to nominate techniques and measures from Table A.4.

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Formal methods	D.28	-	R	R	HR	HR
2	Modelling	Table A.17	R	HR	HR	HR	HR
3	Structured Methodology	D.52	R	HR	HR	HR	HR
4	Modular Approach	D.38	HR	M	M	M	M
5	Components	Table A.20	HR	HR	HR	HR	HR
6	Design and Coding Standards	Table A.12	HR	HR	HR	M	M
7	Analysable Programs	D.2	HR	HR	HR	HR	HR
8	Strongly Typed Programming Language	D.49	R	HR	HR	HR	HR
9	Structured Programming	D.53	R	HR	HR	HR	HR
10	Programming Language	Table A.15	R	HR	HR	HR	HR
11	Language Subset	D.35	-	-	-	HR	HR
12	Object Oriented Programming	Table A.22 D.57	R	R	R	R	R
13	Procedural Programming	D.60	R	HR	HR	HR	HR
14	Metaprogramming	D.59	R	R	R	R	R

"M" The method is mandatory for this SIL.
"HR" The method is highly recommended for this SIL.
"R" The method is recommended for this SIL.
"-" The method has no recommendation for or against its usage for this SIL.

Supported by the LDRA tool suite
Verified by the LDRA tool suite

Figure 5: Mapping the capabilities of the LDRA tool suite to "Table A.4 - Software design and implementation" from EN 50128:2011RA2

There are several aspects of software design and implementation to be considered, starting with a definition of guidelines associated with how the selected programming language is to be used. Figure 5 is a modified version of Table A.4 reproduced from EN 50128, and it shows appropriate techniques, superimposed here with an illustration of where the LDRA tool suite can confirm compliance, or can assist with the confirmation of compliance.

Table A.4 row 1 highlights the use of formal methods - but it is not always obvious when formal methods are in use. Formal methods in the sense of mathematically-based analysis techniques applied to an appropriate model of complete, or partially complete, software are built into the LDRA tool suite.

Table A.4 row 4 references EN 50128 §D.38 which describes a modular approach to software design, which aims to limit the complexity of software by means of decomposition into small comprehensible

parts. Row 7 references EN 50128 §D.2 which is concerned with ensuring that programs are easy for static analysis tools (including the LDRA tool suite) to cope with by adhering to the rules of structured programming.

Table A.4 row 9 references EN 50128 §D.53 which discusses the largely complementary concept of structured programming. The TBvision component of the LDRA tool suite supports these activities through quality metrics - measuring the complexity, loop nesting depth, procedure count, and other parameters with reference to configurable threshold values.

The terms “coding standards”, “language subsets”, and “coding rules” are often used interchangeably, with coding style guides being a related topic. Table A.4 row 11 references §D.35, and row 6 references table A.12 in this regard (Figure 6).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Coding Standard	D.15	HR	HR	HR	M	M
2	Coding Style guide	D.15	HR	HR	HR	HR	HR
3	No Dynamic Objects	D.15	-	R	R	HR	HR
4	No Dynamic Variables	D.15	-	R	R	HR	HR
5	Limited use of Pointers	D.15	-	R	R	R	R
6	Limited use of Recursion	D.15	-	R	R	HR	HR
7	No Unconditional Jumps	D.15	-	HR	HR	HR	HR
8	Limited Size and Complexity of Functions, Subroutines and Methods	D.38	HR	HR	HR	HR	HR
9	Entry/Exit Point Strategy for Functions, Subroutines and Methods	D.38	R	HR	HR	HR	HR
10	Limited Number of Subroutine Parameters	D.38	R	R	R	R	R
11	Limited use of Global Variables	D.38	HR	HR	HR	M	M

“M” The method is mandatory for this SIL.
“HR” The method is highly recommended for this SIL.
“R” The method is recommended for this SIL.
“-” The method has no recommendation for or against its usage for this SIL.

Verified by the LDRA tool suite

Figure 6: Mapping the capabilities of the LDRA tool suite to “Table A.12 — Coding standards” from EN 50128:2011RA2

All of these guidelines can be justified on the basis that they make the resulting code more reliable, less prone to error, easier to test, and/or easier to maintain. For example:

- Limited complexity (Table A.12, row 8) is important because complex code is less easy to read and maintain, and hence more susceptible to error. For low complexity to be enforced, it needs to be quantified and “pass/fail” criteria established.
- Coding standards/Language subsets (Table A.4, row 11 and Table A.12, row 1) such as MISRA C restrict the use of a programming language to those elements known to be least susceptible to causing problems.
- Use of style guides (Table A.12, row 2) ensures that the code has a consistent appearance no matter who has written it. That makes it much easier to maintain or modify, which in turn makes it less prone to error.

The TBvision component of the LDRA tool suite can be used to verify adherence to the coding rules specified by the coding standard, style guide, and/or language subset. The traditional approach to enforcing adherence to such guidelines would be to use peer code reviews. These may well still have a place in the development process – they can be very useful as an aid to learning between team members, for example – but automating the more tedious checks is far more efficient and less prone to error (Figure 7).

Language subsets

There are many language subsets (or “coding standards”) each with differing attributes but nevertheless with strong similarities, especially when referencing the same language. The most popular standards include:

C

MISRA C
CERT C
CWE

C++

MISRA C++
JSF++ AV
HIC++
CERT C++

Java

CWE
CERT J

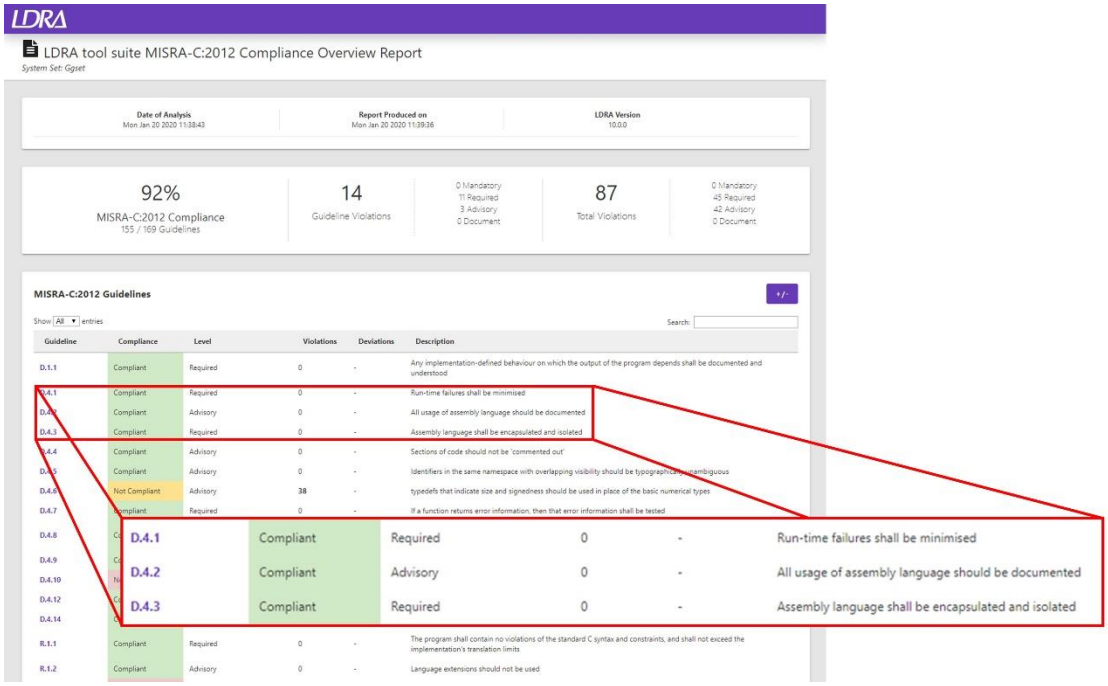

Figure 7: Reviewing coding rules and guidelines in the LDRA tool suite

There are many different sets of coding rules available (sidebar), and even supposing a particular set is chosen as a basis it is entirely permissible to manipulate, adjust and add to it to make it more appropriate for a particular application. Clearly, if a tool is to be useful in such circumstances then it too must be able to accommodate these adjustments.

Table A.5 in EN 50128 provides an overview of applicable verification and testing techniques (Figure 8).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Formal Proof	D.29	-	R	R	HR	HR
2	Static Analysis	Table A.19	-	HR	HR	HR	HR
3	Dynamic Analysis and Testing	Table A.13	-	HR	HR	HR	HR
4	Metrics	D.37	-	R	R	R	R
5	Traceability	D.58	R	HR	HR	M	M
6	Software Error Effect Analysis	D.25	-	R	R	HR	HR
7	Test Coverage for Code	Table A.21	R	HR	HR	HR	HR
8	Functional/ Black-box Testing	Table A.14	HR	HR	HR	M	M
9	Performance Testing	Table A.18	-	HR	HR	HR	HR
10	Interface Testing	D.34	HR	HR	HR	HR	HR

"M" The method is mandatory for this SIL.
"HR" The method is highly recommended for this SIL.
"R" The method is recommended for this SIL.
"-" The method has no recommendation for or against its usage for this SIL.

Supported by the LDRA tool suite
Verified by the LDRA tool suite

Figure 8: Mapping the capabilities of the LDRA tool suite to "Table A.5 — Verification and testing" from EN 50128:2011+A2

Table A.5 row 4 references EN 50128 §D.37 which discusses different quality metrics. TBvision's quality review functionality metrics including clarity, maintainability, testability, complexity (knots, cyclomatic complexity, essential knots, essential cyclomatic complexity), Halstead's metrics (total operators, total operands, unique operators, unique operands vocabulary, length, and volume) and Loop/Interval Analysis (number of loops, depth of loop nesting, number of order 1 intervals, and maximum interval nesting)

Table A.5 row 4 references EN 50128 §D.58. Traceability is a recurring theme throughout the standard and the TBmanager component of the LDRA tool suite is discussed in this regard later in this paper.

Static analysis involves examining source code without executing the program it is part of.

Dynamic analysis involves executing programs, whether in part or as complete applications, on a real or virtual processor.

Both static and dynamic analysis can be undertaken with or without the assistance of automated tools.

Table A.5 references several other, more detailed tables in a hierarchical manner. It highlights static analysis as an applicable technique (row 2). Table A.19 in EN 51028 refers (Figure 9).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Boundary Value Analysis	D.4	–	R	R	HR	HR
2	Checklists	D.7	–	R	R	R	R
3	Control Flow Analysis	D.8	–	HR	HR	HR	HR
4	Data Flow Analysis	D.10	–	HR	HR	HR	HR
5	Error Guessing	D.20	–	R	R	R	R
6	Walkthroughs/Design Reviews	D.56	HR	HR	HR	HR	HR

"M" The method is mandatory for this SIL.
"HR" The method is highly recommended for this SIL.
"R" The method is recommended for this SIL.
"-" The method has no recommendation for or against its usage for this SIL.
Supported by the LDRA tool suite
Verified by the LDRA tool suite

Figure 9: Mapping the capabilities of the LDRA tool suite to "Table A.19 — Static analysis" from EN 50128:2011+A2

Static analysis helps to detect code that may lead to run time errors and general programming errors by ensuring adherence to the coding standards mentioned earlier. Boundary value analysis (Table A.19, row 1) is further supported using dynamic analysis (below).

Checklists (Table A19, row 2) are provided by the TBmanager component of the LDRA tool suite, and to some degree are automatically maintained by it (see "Managing the documentation" below).

The TBvision component offers the control and data flow techniques described in Table A19, rows 3 and 4. As shown in Figure 4, the tool derives the relationships between code components and represent data and control flow graphically.

Again, these representations can be supplemented through dynamic analysis using images superimposed with coverage information collated when the code is executed during unit or system test. Table A.5, row 3 also references dynamic analysis which is detailed in Table A.13 (Figure 10).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Test Case Execution from Boundary Value Analysis	D.4	-	HR	HR	HR	HR
2	Test Case Execution from Error Guessing	D.20	R	R	R	R	R
3	Test Case Execution from Error Seeding	D.21	-	R	R	R	R
4	Performance Modelling	D.39	-	R	R	HR	HR
5	Equivalence Classes and Input Partition Testing	D.18	-	R	R	HR	HR
6	Structure-Based Testing	D.50	-	R	R	HR	HR

"M" The method is mandatory for this SIL.
"HR" The method is highly recommended for this SIL.
"R" The method is recommended for this SIL.
"-" The method has no recommendation for or against its usage for this SIL.
Verified by the LDRA tool suite

Figure 10: Mapping the capabilities of the LDRA tool suite to "Table A.13 — Dynamic analysis" from EN 50128:2011+A2

The TBrun component of the LDRA tool suite supports the techniques highlighted in Figure 10 (Table 13 rows 1,2,3,5 and 6). TBrun automatically generates test drivers and harnesses (wrapper code), enables tests to be easily and efficiently executed, and stores both test data and results (Figure 11). These tests can be automatically regressed. The test data maintenance process is streamlined through the automatic detection of changes in source code, prompting repeats of tests as necessary.

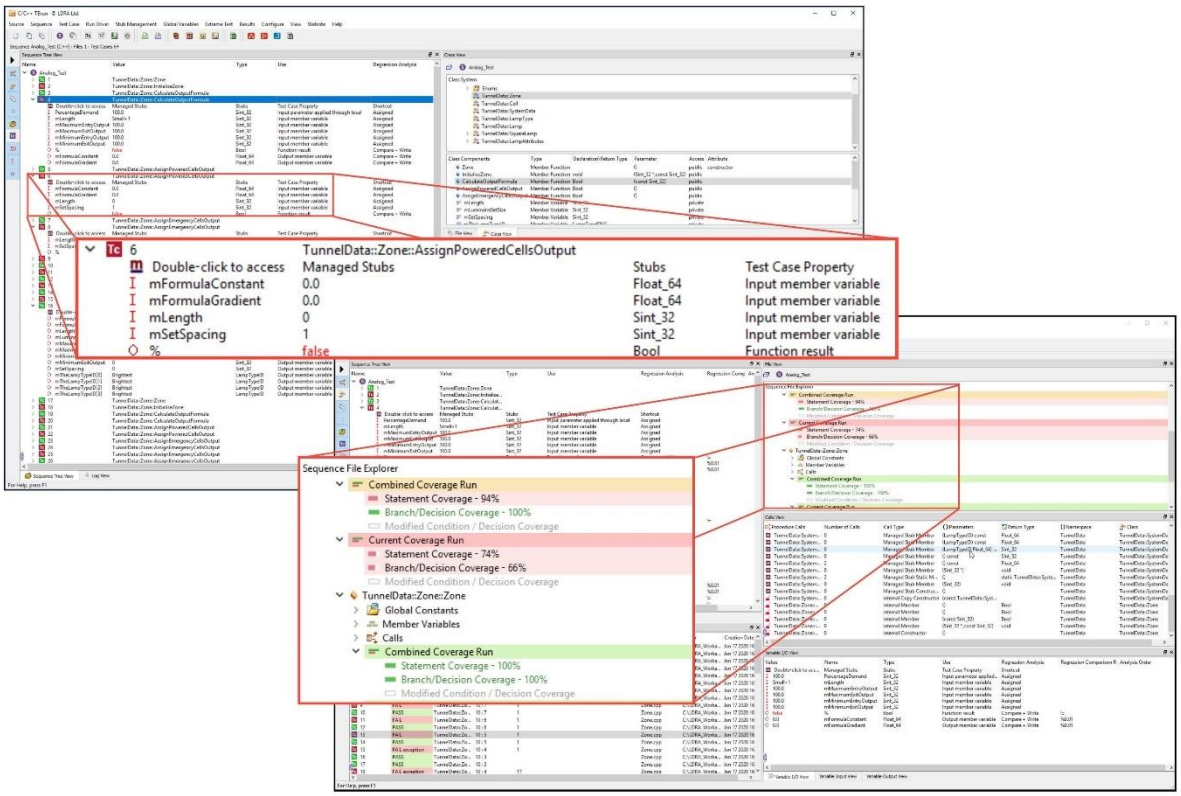

Figure 11:

The TBextreme option supplements TBrun, offering the option to automatically generate test cases, the nature of which is user configurable (Figure 12).

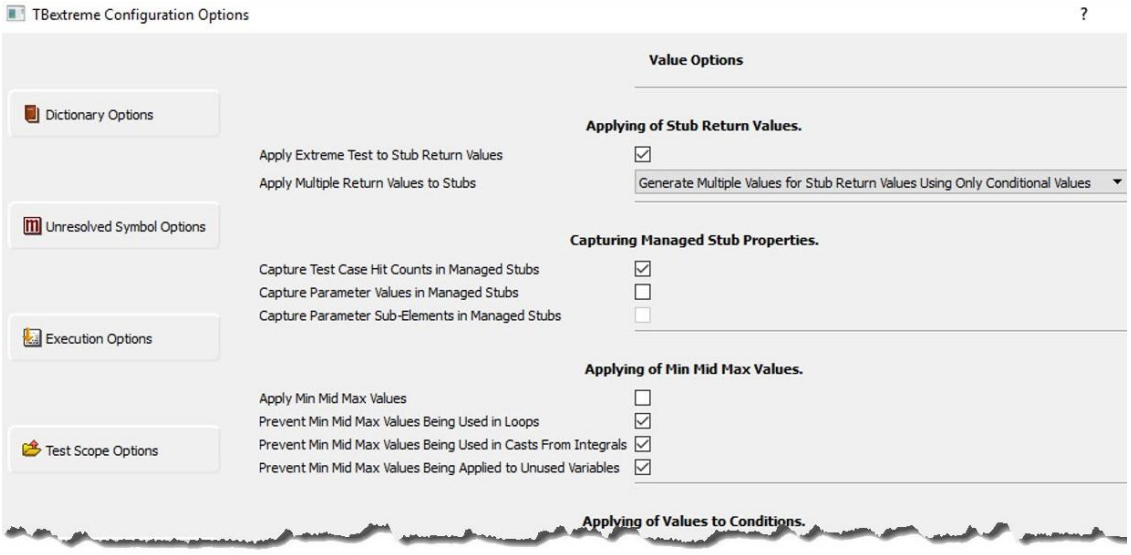

Figure 12: TBextreme is highly configurable

Table A.13 row 1 mentions boundary value analysis which is used to detect and remove software errors occurring at parameter limits or boundaries. Here, the TBextreme option can be configured to create appropriate tests using information gleaned from the static analysis of the code.

Table A.13 row 6 references structure-based testing, which is closely related to the test coverage for code highlighted in Table A.5 row 7. The latter entry is cross referenced to Table A.21 (Figure 13) which details techniques that can all be performed using the LDRA tool suite to determine the required coverage information as specified by the standard - statement, branch/decision, MC/ DC (Modified Coverage/Decision Coverage), procedure/call coverage, etc.

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Statement	D.50	R	HR	HR	HR	HR
2	Branch	D.50	-	R	R	HR	HR
3	Compound Condition	D.50	-	R	R	HR	HR
4	Data flow	D.50	-	R	R	HR	HR
5	Path	D.50	-	R	R	HR	HR
"M" The method is mandatory for this SIL. "HR" The method is highly recommended for this SIL. "R" The method is recommended for this SIL. "-" The method has no recommendation for or against its usage for this SIL. Verified by the LDRA tool suite							

Figure 13: Mapping the capabilities of the LDRA tool suite to "Table A.21 — Test coverage for code" from EN 50128:2011+A2

Figure 14 illustrates some of the many devices used within the LDRA tool suite to illustrate that coverage.

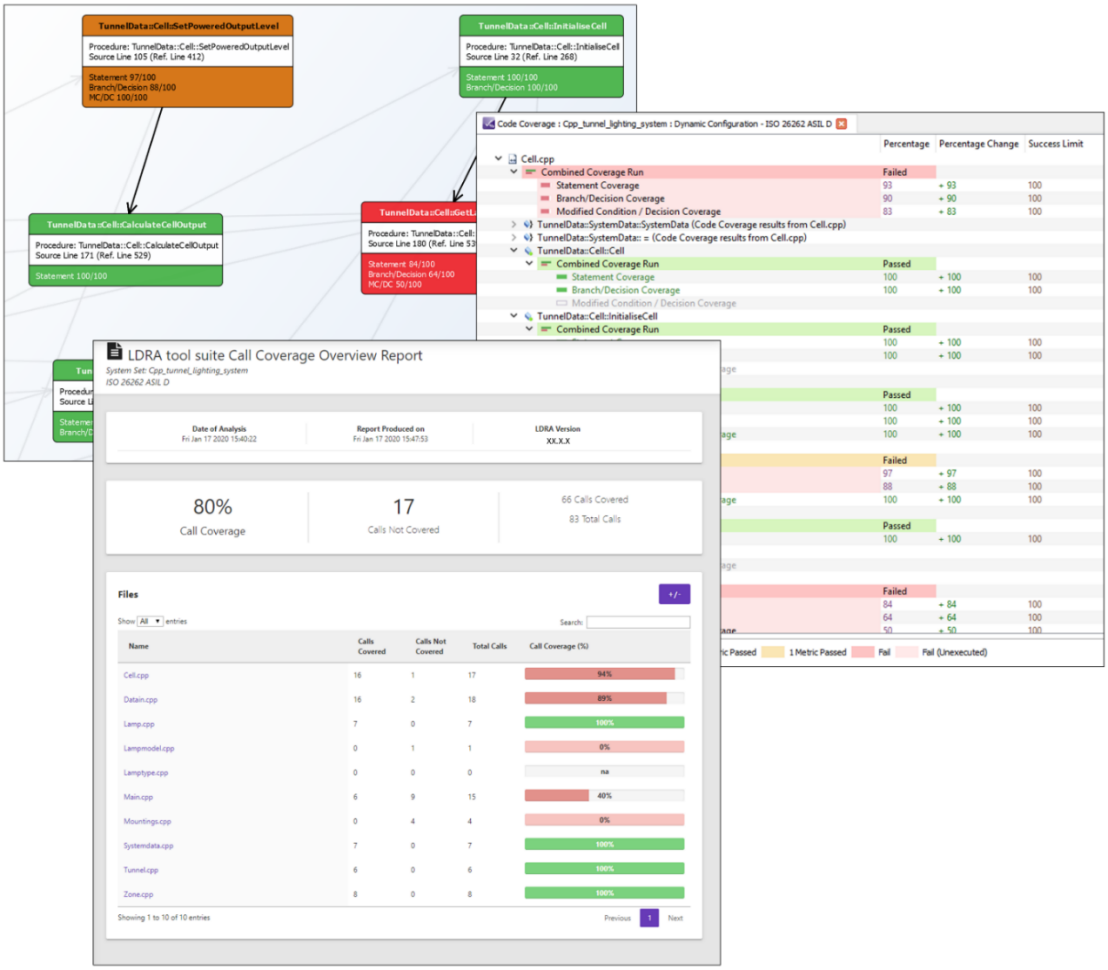

Figure 14: Representations of code coverage in the LDRA tool suite

Metrics linked to the execution of statements, branch/decisions, compound conditions (MC/DC), LCSAJ, and basic path can all be checked as referenced by EN 50128 §D.50. Table A.5 row 6 references Functional/Black box testing and references table 14 (Figure 15).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Test Case Execution from Cause Consequence Diagrams	D.6	-	-	-	R	HR
2	Prototyping/Animation	D.43	-	-	-	R	R
3	Boundary Value Analysis	D.4	R	HR	HR	HR	HR
4	Equivalence Classes and Input Partition Testing	D.18	R	HR	HR	HR	HR
5	Process Simulation	D.42	R	R	R	HR	HR
"M" The method is mandatory for this SIL. "HR" The method is highly recommended for this SIL. "R" The method is recommended for this SIL. "-" The method has no recommendation for or against its usage for this SIL. Verified by the LDRA tool suite							

Figure 15: Mapping the capabilities of the LDRA tool suite to "Table A.14 – Functional/black-box testing" from EN 50128:2011+A2

As previously mentioned, TBrun can be used to perform boundary value analysis (Table A.14, row 3). The integration of the LDRA tool suite with modelling tools such as IBM Engineering Systems Design Rhapsody, and Ansys SCADE provides the capability to create test cases from sequence diagrams, classes and packages (Table A.14, row 1).

Software component implementation and testing (EN 50128 §7.5)

The EN 50128 V-model shows implementation and testing as separated elements, but both are handled as one phase in section 7.5 of the EN 50128 standard as reflected in the objectives*: "To achieve software which is analysable, testable, verifiable and maintainable. Component testing is also included in this phase"* (EN 50128 §7.5.1.1). The V-model also suggests that two design and test documents are to be produced, along with source code and its supporting documentation. The documents are described in the following EN 5012X sections:

EN 50128 §7.5.4.1 to §7.5.4.4 Software Source Code and Supporting Documentation
EN 50128 §7.4.4.5 to §7.3.4.7 Software Component Test Report
EN 50128 §7.5.4.9 Software Source Code Verification Report

The software requirements verification report is written at the end of the phase on the basis of the other listed documents.

Although this section of the standard represents a significant part of the development effort, it is quite short because it describes the implementation of the techniques that are specified at length earlier in the process. Most of the artefacts required by this section in EN 50128 §7.5.4.10 – including evidence of "*The correct use of the chosen techniques and measures from table A.4*", and "*the correct application of coding standards*" are generated by the tool suite as those measures are applied. The *"requirements for readability and traceability"* are supported by the TBmanager component of the LDRA tool suite, as discussed in the section "Managing the documentation" below.

Software integration phase (EN 50128 §7.6)

The software integration phase as represented in the EN 50128 V-model suggests that three design and test documents are to be produced, as described in the following EN 5012X sections and in accordance with the Software/Hardware Integration Test Specification and the Software Integration Test Specification:

EN 50128 §7.6.4.3 to §7.6.4.6 Software Integration Test Report
EN 50128 §7.6.4.7 to §7.6.4.10 Software/Hardware Integration Test Report
EN 50128 §7.6.4.11 to §7.6.4.13 Software Integration Verification Report

The Software Integration Verification Report is written at the end of the phase on the basis of the other listed documents.

The objectives of the phase are to *"carry out software and software/hardware integration"* and to show that they *"perform their intended functions"* (EN 50128 §7.6.1). Much of the integration phase therefore concerns the implementation of principles laid out in previous sections. Table A.6 is also referenced in the context of both software integration, and software/hardware integration (Figure 16).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Functional and Black-Box Testing	Table A.14	HR	HR	HR	HR	HR
2	Performance Testing	Table A.18	–	R	R	HR	HR

"M" The method is mandatory for this SIL.
"HR" The method is highly recommended for this SIL.
"R" The method is recommended for this SIL.
"-" The method has no recommendation for or against its usage for this SIL.
Verified by the LDRA tool suite
Supported by the LDRA tool suite

Figure 16: Mapping the capabilities of the LDRA tool suite to "Table A.6 — Integration" from EN 50128:2011+A2

The TBrun component of the LDRA tool suite supports software integration. Software functions and procedures can be tested in isolation of as part of a call tree, meaning that the tests used during unit test can be re-run during integration testing to show that their functionality remains consistently correct in the wider context. It supports both black-box testing (when the functionality alone is checked – Table A.6 row 1) and white-box testing (when code coverage information is also collated).

The tool suite also supports testing on target hardware. One common approach that limits demands on often scarce target hardware is to first complete software integration testing using a simulator, and then complete software/hardware integration testing on the target. TBrun supports timing analysis which can be performed to check whether the function is executed within a given period of time (Table A.6 row 2), and the LDRA tool suite also supports assembly code coverage on a multitude of different processors.

Software validation phase (EN 50128 §7.7)

The software validation phase as represented in the EN 50128 V-model suggests that three design and test documents are to be produced, as described in the following EN 5012X sections and in accordance with the Overall Software Test Specification and other supporting documentation.

EN 50128 §7.6.4.3 to §7.6.4.6 Overall Software Test Report
EN 50128 §7.6.4.11 to §7.6.4.13 Software Validation Report
EN 50128 §7.7.4.2 Release Note

The Software Integration Verification Report is written at the end of the phase on the basis of the other listed documents.

The objectives of the phase are to "*ensure compliance with the Software Requirements Specification"* and to *"check whether [the integrated software and hardware] is fit for its intended purpose."* Arguably the software validation phase is simply the last step in the integration phase, where all software and hardware components have been integrated into a complete system. It does however demand more complete tracing back to requirements and intermediate evidential artefacts, as reflected in the six input documents. Automating that traceability will help to alleviate the project management burden, as discussed below.

Managing the documentation

The engineering justification for the processes outlined in the EN 5012X standards is clear. In common with other functional safety standards, it represents the simple application of best practices along with the collation of evidence that those practices have been correctly applied. Such an approach has proven successful in many safety critical sectors, including this one.

However, it is also clear that there are many artefacts to collate, cross-reference and keep track of. Ensuring that these are permanently up-to-date can be a logistical nightmare in the face of changing customer requirements, failed tests and engineering oversights – all of which are real-life features of complex projects. Supporting that project management overhead by means of automation can make that process both more effective, and much less time consuming.

Software assessment phase (EN 50128 §6.4)

The software assessment phase puts into sharp focus the need to maintain evidential artefacts and their traceability at all times. It is shown detached from the V-model, and towards the end of it in parallel with the validation and maintenance phases. However, EN 50128 §6.4.4.3 makes clear that "*The Assessor shall have access to all project-related documentation throughout the development process."* Although the software assessment activity will clearly not concluded until the development lifecycle, it is therefore intended to be a continuous process.

EN 50128 §6.4.4.9 underlines that point. *"The Assessor shall assess if an appropriate set of techniques from Annex A, suitable for the intended development, has been selected and applied… the Assessor shall consider the extent to which each technique from Annex A is applied…and also look for evidence that it is properly applied".*

Bidirectional traceability

Traceability is referenced throughout the EN 5012X series of standards, and the repeated requirement for evidence of it (EN 50128 §5.3.2.7, §6.5.4.14, §7.5.4.10, §D.58 etc.) reinforces its role as a key objective of the standard.

It is easy to mistake the notion of traceability as a one-way street, assuming that as long as the requirements are all shown to be implemented and tested in the code the objective of the standard is satisfied. However, it is just as important to show that there is no code present that satisfies no requirements. Such code may be the result of feature creep, of code that was created before an alternative implementation was selected, or even of a deliberately implanted “back door” method added by the unscrupulous. In any event, providing evidence that there is no such code is important.

In theory, this so-called “bidirectional traceability” is easy to achieve. If the exact sequence of the V-model is adhered to, then the requirements will never change and tests will never throw up a problem. Challenges do arise during the course of a project and traceability between all project phases can become challenging to achieve. Automating that traceability and highlighting inconsistencies helps to address that challenge (Figure 17).

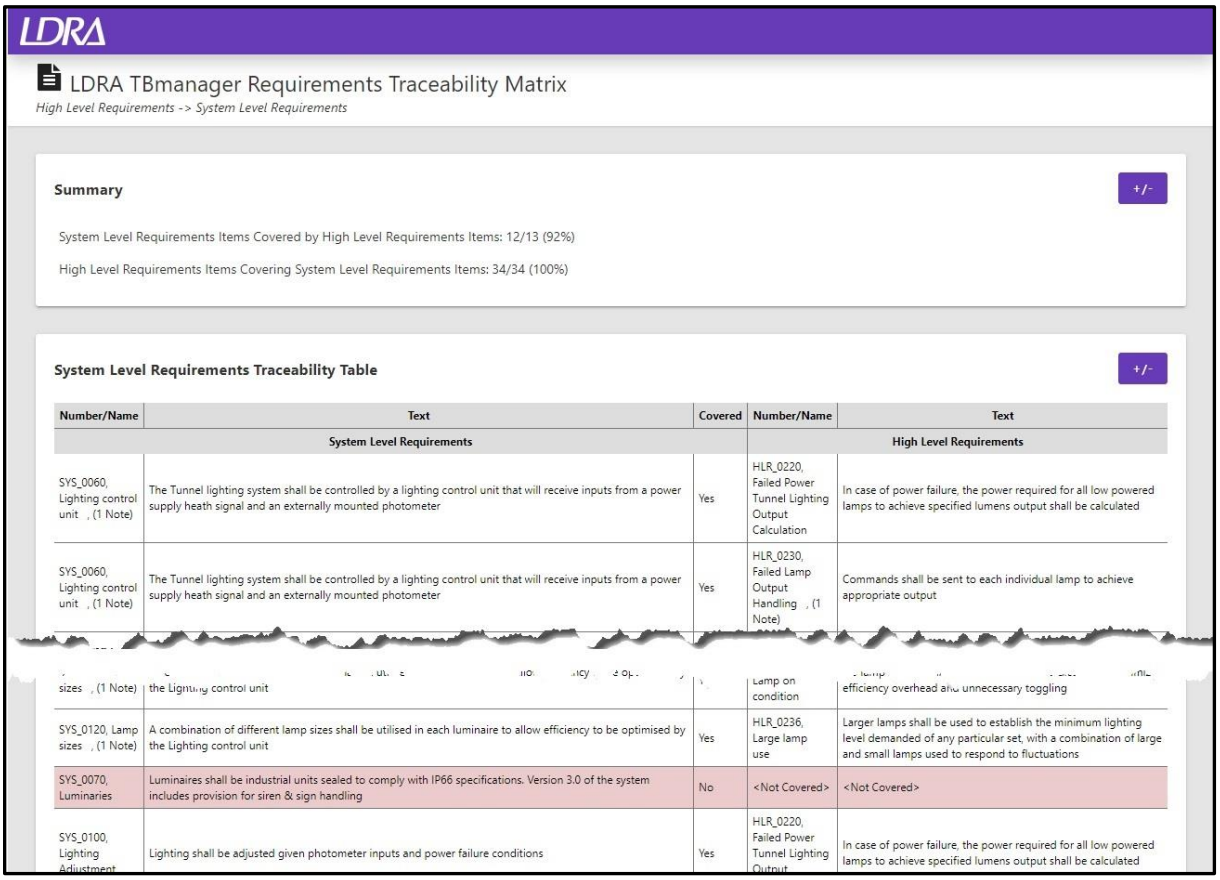

Figure 17: Highlighting missing coverage of requirements with the LDRA tool suite.

Consider what happens if a software component implementation test fails.

- It may fail because there is a contradiction in the requirements. If that is the case, the requirements will need to change. But what other parts of the software are affected by that?
- It may fail because there is a coding error. If that is the case, then it will need to be corrected. But what other software units were dependent on the modified code? What if they were dependent on an incorrect output?

- It may fail because the requirements have an incorrect specification for the test parameters. That means a requirement change. But have there been unit tests which are compromised because these parameters were wrong?

These scenarios can quickly lead to situations where the traceability between the products of software development falls down. While it is possible to perform the tasks of maintaining it manually, automation is likely to help a great deal.

Software unit design can take many forms and can leverage natural language or model-based approaches. Either way, these design elements need to be bidirectionally traceable to both software safety requirements and the software architecture. The software units must then be implemented as specified and then be traceable to their design specification (Figure 18).

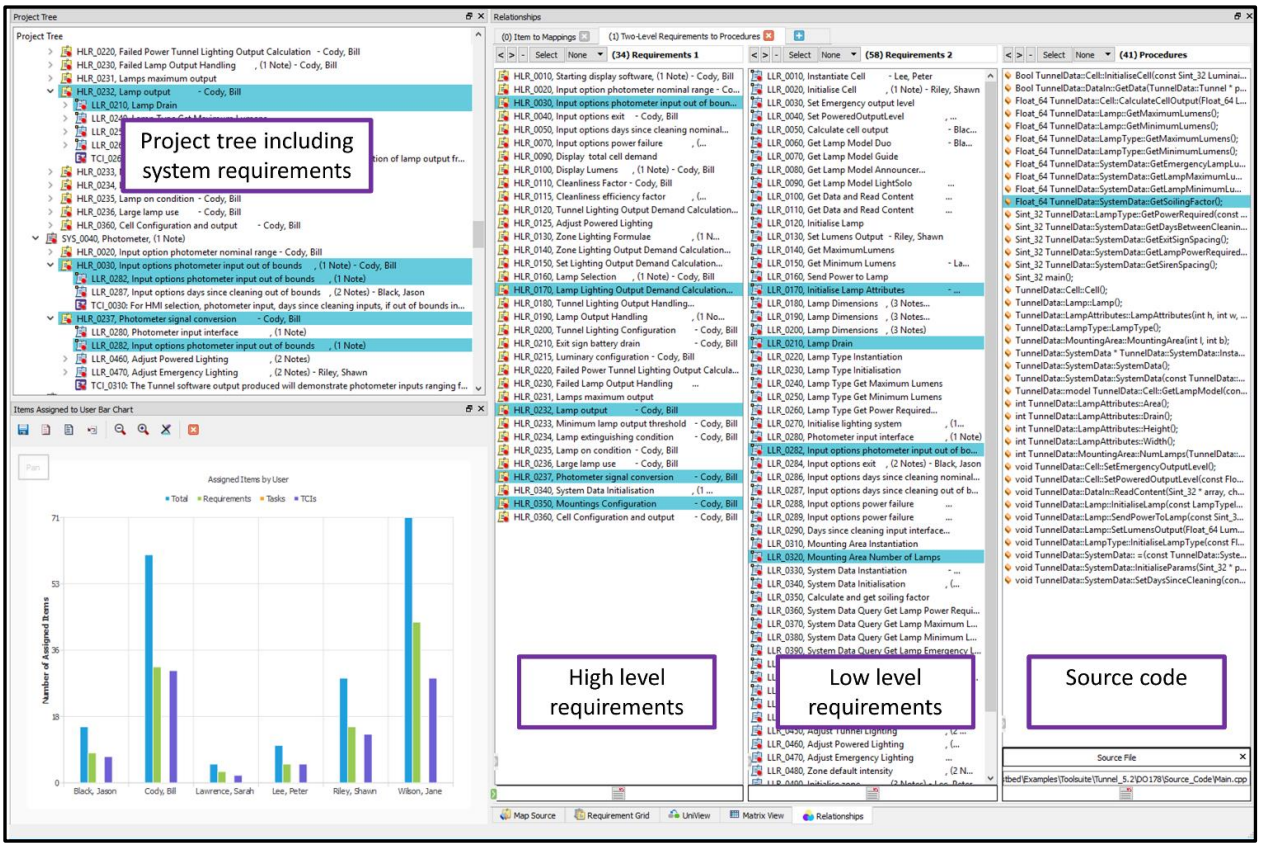

Figure 18: Traceability in the LDRA tool suite. The example detailed design is linked upstream to software safety requirements and downstream to software units.

Figure 19 shows how to establish a traceability policy can be established between requirements and tests cases of different scopes, which allows test coverage to be assessed. Test cases are reviewed and developed based on requirements, and gaps in requirements coverage can be quickly assessed and filled.

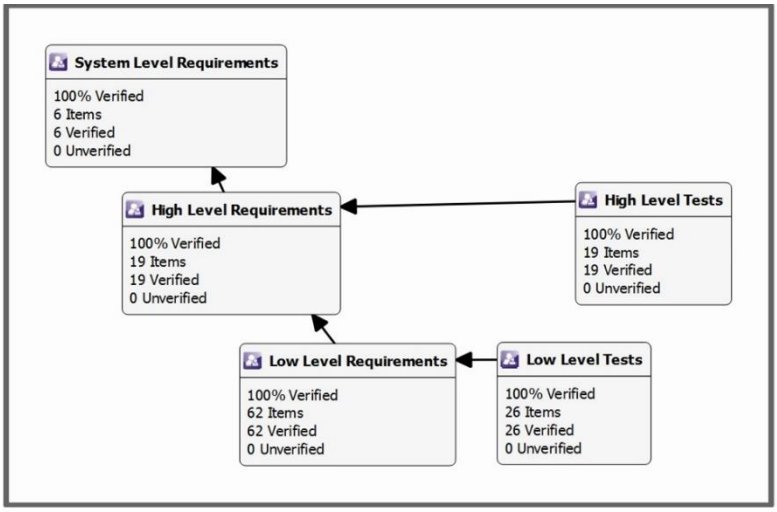

Figure 19: Performing requirement based testing. Test cases are linked to requirements and executed within the LDRA tool suite

Conversely, bidirectional analysis can also determine test cases that are not linked to requirements, and prompt for their correction. Additionally, it provides a mechanism for impact analysis with reference to changes in requirements, tests, or code, and identifies where regression testing needs to be targeted in order to minimize rework overhead. It also provides a mechanism for the generation of artefacts such as traceability matrices to present evidence of compliance to EN 5012X.For example, suppose a traceability matrix similar to that in Figure 17 displays the relationship between high-level requirements and functional test cases and that there is a requirement which has no associated test case. This would immediately highlight the fact that the objective of test coverage of high-level requirements is unfulfilled.

In practise, initial structural coverage is often accrued as part of a holistic process from the execution of functional tests on instrumented code leaving unexecuted portions of code which require further analysis, ultimately resulting in the addition or modifications of test cases, changes to requirements, and/or the removal of dead code. An iterative “review, correct and analyse” sequence is typically needed to ensure that design specifications are satisfied.

Keeping track of evidential artefacts

Although TBmanager is clearly adept at tracing the status of tests process supported by the LDRA tool suite, its usefulness does not end there. It also supports the tracing of artefacts generated by other means, and provides traceability to the objectives of the EN 5012X standards (Figure 20).

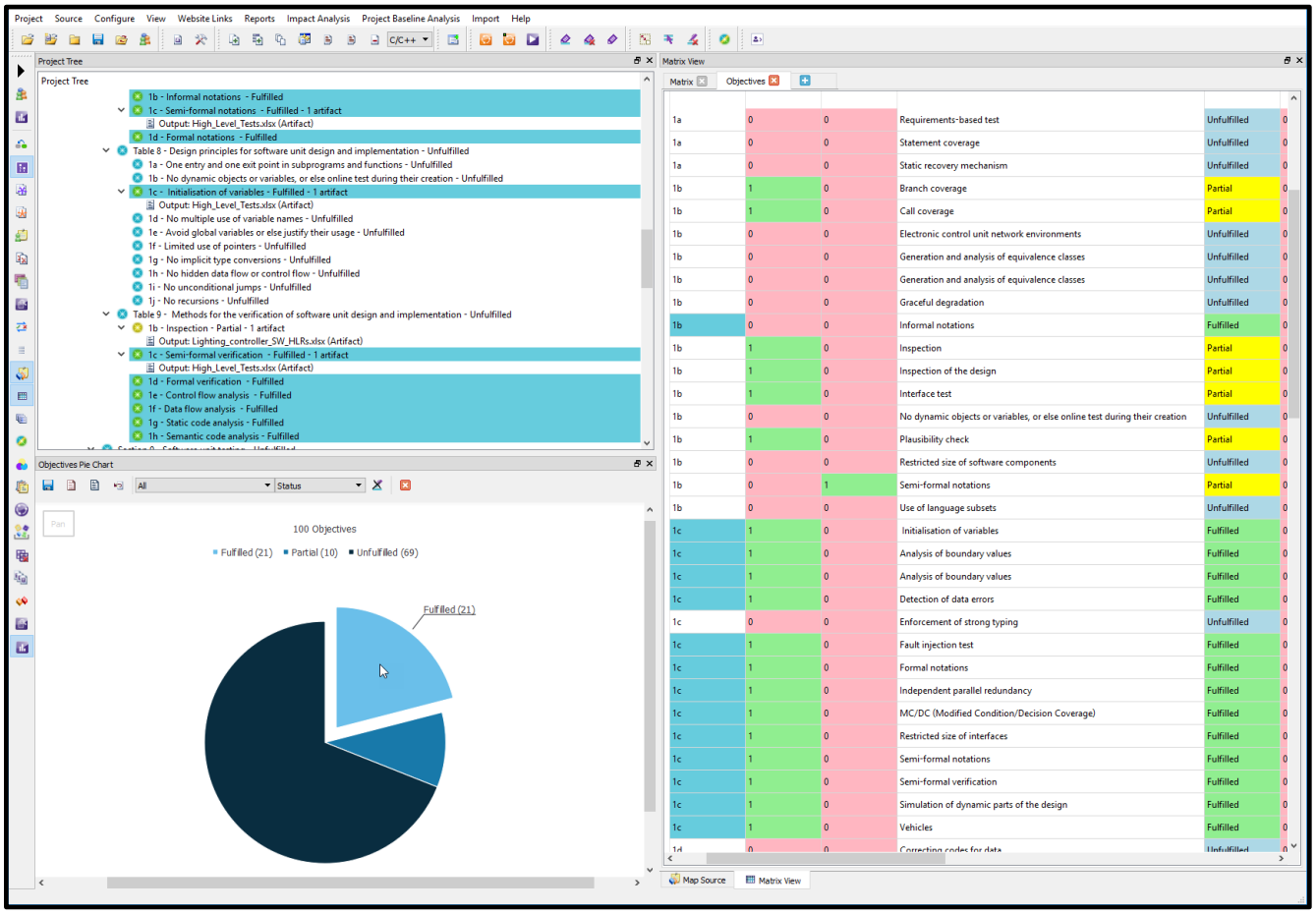

Figure 20: The TBmanager component of the LDRA tool suite also automates traceability to standard objectives

The net result is a window into the world of the project as it evolves, with cross references to standard objectives, project requirements, designs, and code all maintained. Such information is available to view on a context-sensitive basis such that (for example) the engineering information required by a test engineer is not confused by data designed to fulfil the needs of a project manager, and vice versa.

Software maintenance phase (EN 50128 §9.2)

The software maintenance phase as represented in the EN 50128 V-model suggests that associated documents are to be produced, as described in the following EN 5012X sections. All documents created during the design and development process are cited as relevant.

EN 50128 §9.2.4.5 and §9.2.4.6 Software Maintenance Plan
EN 50128 §9.2.4.9 and §9.2.4.10 Software Change Records
EN 50128 §9.2.4.7 and §9.2.4.8 Software Maintenance Records
EN 50128 §9.2.4.11 to §9.2.4.14 Software Maintenance Verification Report

The objective of this phase is to *"To ensure that the software performs as required, preserving the required software safety integrity level and dependability when making corrections, enhancements or adaptations to the software itself."*

EN 50128 cross references to several other standards in this regard. It references "'*Phase 11: Operation, maintenance and performance monitoring*' *in EN 50126-1*" (EN 50128 §9.2.11), requires that *"Maintenance shall be carried out in accordance with the guidelines contained in IOS/IEC*

90003[18]*"* (EN 50128 §9.2.4.3), and demands that the "*ISO/IEC 25000*[19] *series shall also be employed in order to implement and verify a minimum level of maintainability*" (EN 50128 §9.2.4.4).

	TECHNIQUE/MEASURE	REF.	SIL				
			Basic Integrity	1	2	3	4
1	Impact Analysis	D.32	R	HR	HR	M	M
2	Data Recording and Analysis	D.12	HR	HR	HR	M	M

"M" The method is mandatory for this SIL.
"HR" The method is highly recommended for this SIL.
"R" The method is recommended for this SIL.
"-" The method has no recommendation for or against its usage for this SIL.

Verified by the LDRA tool suite

Figure 21: Mapping the capabilities of the LDRA tool suite to "Table A.10 —Software maintenance" from EN 50128:2011+A2

Table A.10 refers to software maintenance (Figure 21). Impact analysis aims to determine the extent to which a change or an enhancement to a software system will impact upon the existing system. A decision is required concerning the verification of the modified software system. This depends on the number of software modules affected, the criticality of the affected software modules and the nature of the change. Possible decisions are:

- Only the changed software module is re-verified
- All affected software modules are re-verified
- The complete system is re-verified

The analysis facilities provided by the TBmanager of the LDRA tool suite throughout the development lifecycle are arguably even more valuable during maintenance.

The nature of connected systems in particular has had a subtle but significant impact on the demands of software maintenance, especially with regards to incident response which is certainly covered by the objectives of EN 5012X wherever there are safety implications. Connectivity has the potential to have a disproportionate impact on the nature of many development teams, who are brought together to work on a particular project and go their separate ways after its conclusion.

In an environment where it is necessary to defend not only the world from the system but also the system from the world, traditional truths represented by the notions of infant mortality and the bathtub curve may no longer apply. There may be no incident of note for years after the launch of a product or feature – or to put it another way, perhaps not until all the developers with intimate knowledge of it have moved on. The availability of a tool to provide instant impact analysis, to assess the scope of necessary changes and retest, and to automatically implement those retests would clearly be invaluable in responding to cyberattack.

Support tools (EN 50128 §6.7)

EN 50128 §6.7.1.1 requires that *"evidence is provided that potential failures of tools do not adversely affect the integrated toolset output in a safety related manner that is undetected by technical and/or organisational measures outside the tool."*

[18] ISO/IEC 90003:2014, Software engineering – Guidelines for the application of ISO 9001 to computer software
[19] ISO/IEC 25000 series, Systems and software engineering – Systems and software Quality Requirements and Evaluation

The standard defines a mechanism to provide evidence that the software tool chain can be relied upon. The required level of confidence in a software tool depends upon the circumstances of its deployment, with particular reference to the possibility that the malfunctioning software tool and its corresponding erroneous output can introduce or fail to detect errors in a safety-related development, and the confidence in preventing or detecting such errors in its corresponding output.

Tools are categorised into one of three categories. The LDRA tool suite is placed in the T2 (Tool Class 2) category because it *"supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software"* (EN 50128 §3.1.42). That differentiates tools like the LDRA tool suite from (say) auto code generators which have the potential to introduce errors into an application.

Placement in that category obliges developers to comply with EN 50128 §6.7.4.1, §6.7.4.2, §6.7.4.3, §6.7.4.10, and §6.7.4.11. These require that:

- *"Software tools shall be selected as a coherent part of the software development activities"* (EN 50128 §6.7.4.1)
- "*The selection of the tools in classes T2 and T3 shall be justified"* (EN 50128 §6.7.4.2)
- "*All tools in classes T2 and T3 shall have a specification or manual which clearly defines the behaviour of the tool and any instructions or constraints on its use"* (EN 50128 §6.7.4.3)
- *"Configuration management shall ensure that for tools in classes T2 and T3, only justified versions are used"* (EN 50128 §6.7.4.10)
- *"Each new version of a tool that is used shall be justified"* (EN 50128 §6.7.4.11)

The LDRA tool suite has been certified as "fulfilling the requirements of support tools according to EN 50128" by two separate TÜV organisations (Figure 22). The existence of such a certificate is sufficient to establish confidence in the tool, saving considerable overhead in establishing that independently.

Figure 22: Evidence of TÜV approved qualification of the LDRA tool suite

Conclusions

The explosion in the quantity and complexity of the embedded software in Guided Transport Systems is well documented. The EN 5012X standards in their current form fine tunes the sound foundations established by the earlier versions, with the EN 50128 amendments bringing that standard into line with the rest of the set.

The EN 5012X standards do not directly address cybersecurity which is increasingly a concern for connected systems, and their use in parallel with the IEC 62334-4-1 standard is a popular approach to addressing that concern.

There is clearly an incentive to minimise the SIL applied to each element of an EN 5012X compliant system, because of the higher cost involved in achieving higher SILs. However, the whole principle of the assignment of SILs for various systems implies an assumption of separation, such that the most critical systems cannot be compromised by less critical functionality elsewhere. For a connected system to be considered safe and compliant with the principles of EN 5012X, it is therefore imperative that attack surfaces are minimized, and that separation between systems is optimized.

With that assurance in place, the task of maintaining the appropriate documentation at all times for each part of the system can be a daunting one. Automating not only the test processes but also the artefacts produced by them and other lifecycle activities can be of considerable help in alleviating the project management overhead of compliance. Tools such as those provided by LDRA are well proven in other safety critical sectors, making them not only ideally placed to further optimize the route to compliance, but also better established than the standard itself!

Works cited

EN 50126-1:2017 Railway Applications. The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS). Generic RAMS Process

EN 50126-2:2017 Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 2: Systems Approach to Safety

EN 50128:2011+A2:2020 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems

EN 50129:2017 Railway applications. Communication, signalling and processing systems. Safety related electronic systems for signalling

IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems

IEC 62278:2002 Railway applications - Specification and demonstration of reliability, availability, maintainability and safety (RAMS)

IEC 62279:2015 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems

IEC 62280:2014 Railway applications - Communication, signalling and processing systems - Safety related communication in transmission systems

RTCA DO-178C Software Considerations in Airborne Systems and Equipment Certification, Prepared by SC-205, December 13, 2011

RTCA DO-278 Guidelines for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance

EN 50126-1:2017 Railway Applications. The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS). Generic RAMS Process

IBM Engineering Requirements Management DOORS Family https://www.ibm.com/uk-en/products/requirements-management

Siemens Polarion https://polarion.plm.automation.siemens.com/

Object Management Group Requirements Interchange Format http://www.omg.org/spec/ReqIF/

IBM Engineering Systems Design Rhapsody https://www.ibm.com/uk-en/products/systems-design-rhapsody

Mathworks Simulink Simulation and Model-Based Design https://www.mathworks.com/products/simulink.html

Ansys SCADE Suite https://www.ansys.com/products/embedded-software/ansys-scade-suite

ISO/IEC 90003:2014, Software engineering – Guidelines for the application of ISO 9001 to computer software

ISO/IEC 25000 series, Systems and software engineering – Systems and software Quality Requirements and Evaluation

CENELEC 50128 and IEC 62279 Standards, Jean-Louis Boulanger, Wiley, ©ISTE Ltd 2015, ISBN 978-1-84821-634-1