LDRA
Software Technology

Technical White Paper

Developing software for household appliances in accordance with IEC 60730

Cost-effective certification for Class B and Class C control software

www.ldra.com
* Registration required to download the document

© 2020 LDRA Ltd. This document is property of LDRA Ltd. Its contents cannot be reproduced, disclosed or utilised without company approval.

Contents

Introduction 3
The need for a process standard 3
Classification of appliance software 5
Application of IEC 60730 process activities 4
Requirements for the architecture (Clause H.11.12.1) 6
Measures to control faults/errors (Clause H.11.12.2) 6
Measures to avoid errors (Clause H.11.12.3) 7
Specification (Clause H.11.12.3.2) 7
Software safety requirements (Clause H.11.12.3.2.1) 7
Software architecture (Clause H.11.12.3.2.2) 9
Module design and coding (Clause H.11.12.3.2.3) 13
Design and coding standards (Clause H.11.12.3.2.4) 15
Testing (Clause H.11.12.3.3) 17
Module design testing (Clause H.11.12.3.3.1) 18
Software integration testing (Clause H.11.12.3.3.2) 21
Software validation (Clause H.11.12.3.3.3) 24
Tools, programming languages, management of software versions, and modification (Clause H.11.12.3.4) 25
Conclusions 27
References 28

Introduction

Household appliances are becoming ever more smart, and the demand for new features is seemingly endless. In particular, the connectivity that brings remote monitoring or control from across the room or across the world is now commonplace.

Generally, and unlike cars, medical equipment or aeroplanes, household appliances do not cause threat to life in the event of malfunction. Development processes have therefore not always been as rigorous as in those safety critical sectors. But extended functionality implies additional complexity, and unhappily that is often accompanied by a less welcome increased rate of failure which is clearly not desirable.

Ensuring the potential failures are within acceptable limits is a challenge if customer safety and security are to be ensured, and brand reputation untarnished. Appropriate measures to ensure safety, security and reliability are required from the outset and throughout the product development lifecycle. There is no better place to seek inspiration that the industries where software has long been a matter of life or death.

The need for a process standard

Process standards have paved the way for the safe development and deployment of electrical, electronic, and programmable electronic systems since the early 1990s. The military standard MIL 498[1] was instrumental in establishing many best practices for the development of safety/mission-related systems in the defence sector and beyond.

The principles laid down by MIL 498 were enhanced in the IEEE 12207[2] standard which defined lifecycle process activities for software and system development in accordance with the systems engineering standard ISO/IEC 15288[3]. In turn, hardware and software functional safety study groups working in the early 1990s drew inspiration from ISO/IEC 15288 to create their own draft document, IEC 1508[4].

Further enhancements to IEC 1508 (including the first definition of the safety lifecycle) ultimately saw the release of the now ubiquitous IEC 61508[5]. Last updated in 2010[6], IEC 61508 is not only referenced in its generic form by developers of functionally safe electrical, electronic, and programmable electronic systems in industrial applications. It has also formed the basis for numerous industry specific standards including variants for the automotive (ISO 26262[7]), medical device (IEC 62304[8]), and nuclear (IEC 60880[9]) sectors.

---

[1] MILITARY STANDARD: SOFTWARE DEVELOPMENT AND DOCUMENTATION (05 DEC 1994)
http://everyspec.com/MIL-STD/MIL-STD-0300-0499/MIL-STD-498_25500/

[2] ISO/IEC 12207:1995. Information technology — Software life cycle processes. July 1995.
https://www.iso.org/standard/21208.html

[3] ISO/IEC 15288:2008 Systems and software engineering — System life cycle processes
https://www.iso.org/standard/43564.html

[4] IEC 1508: Functional Safety: Safety-Related Systems. August 1995.
https://ieeexplore.ieee.org/document/525946

[5] IEC 61508-1:1998 Functional safety of electrical/electronic/programmable electronic safety-related systems
https://webstore.iec.ch/publication/19800

[6] IEC 61508-1:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems
https://www.iecee.org/dyn/www/f?p=106:49:0::::FSP_STD_ID:5515

[7] ISO 26262-1:2011 Road vehicles — Functional safety
https://www.iso.org/standard/43464.html

[8] IEC 62304:2006+AMD1:2015 CSV Consolidated version Medical device software - Software life cycle processes
https://webstore.iec.ch/publication/22794

[9] Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions
https://webstore.iec.ch/publication/3795

LDRA

IEC 60730[10] “Automatic electrical controls” is also a derivative of IEC 61508. It is focused on electrical and electronic controls associated with or used within household appliances - for example, heating and air-conditioning systems. The standard’s scope includes for appliances using electricity, gas, oil, solid fuel, solar thermal energy, or a combination of these. Furthermore, despite the moniker “household appliance”, it also extends to devices used in public spaces including shops, offices, hospitals, farms, and commercial and industrial premises.

Despite the increasing complexity of their products, household appliance developers are required to ensure that the likelihood of injury to persons or damage to property resulting from their use is very low, even in the event of negligence. The primary purpose of IEC 60730 is to define a process that will ensure these aims are met by ensuring the functional safety of these products (sidebar[11]).

Safety is the freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment.

Functional safety is part of the overall safety of a system or piece of equipment and generally focuses on electronics and related software. It looks at aspects of safety that relate to the function of a device or system and ensures that it works correctly in response to commands it receives.

It provides technical guidelines applicable to any manual (see IEC 60335-1[12]) and automatic electrical controls. These can take many forms. For example, they may:

- form part of an appliance,
- be individual controls utilized as a part of a control system, or
- be mechanically integral with multifunctional controls having non-electrical outputs.

It is incumbent upon a manufacturer seeking to be compliant with IEC 60730 to provide adequate information for a control’s suitability to a particular application to be confirmed, and for it to be mounted, used, and tested in an defined manner.

Classification of appliance software

IEC 60730 discusses mechanical, electrical, electronic, environmental endurance, EMC, and abnormal operation for home appliances. For the evaluation of protective measures for fault tolerance and avoidance of hazards, it classifies control functions according to their potential impact in the event of a fault:

- **Class A** - Control functions that are not intended to be relied upon for the equipment’s safety and have no feature that can harm a human being.
  - For example: humidity controls, lighting controls, timers, and switches.
- **Class B** - Control functions that are intended to prevent unsafe operation of the controlled equipment.
  - For example: thermal cut-offs and door locks for laundry machines.
- **Class C** - Control functions that are intended to prevent special hazards.
  - For example: automatic burner controls and thermal cut-outs for closed, unvented water heater systems.

[10] IEC 60730-1:2013 Automatic electrical controls
https://webstore.iec.ch/publication/3117
[11] IEC: Functional Safety
https://basecamp.iec.ch/download/functional-safety-essential-to-overall-safety/
[12] IEC 60335-1:2010 Household and similar electrical appliances - Safety - Part 1: General requirements
https://webstore.iec.ch/publication/1499

Controls for appliances that fall under Class B are typified by those used for washing machines, dishwashers, dryers, refrigerators, freezers, and cookers/stoves, whereas gas-fired controlled dryers and water heaters that might cause an explosion exemplify Class C[13]. Microcontroller units in Class B & C appliances are typically evaluated following IEC 60335-1 Annex R[14], with IEC 60730 Annex H[15] detailing requirements for software based electronic controls.

The latter annex requires that "*Controls using software shall be so constructed that the software does not impair control compliance with the requirements other aspects of the standard. Compliance is checked by the tests for electronic controls in this standard, by inspection … and by examination of the documentation required.*"[16]

In other words, for a control to be compliant, it needs to comply with all aspects of the standard whether software is involved, or not. The extent to which that requirement places an overhead on a software development team depends on the classification of control functions when, as Annex H confirms, "*their integration into the complete safety concept of the appliance shall be taken into account.*"[17]

Application of IEC 60730 process activities

Constructional requirements for control systems are specified in Clause 11 of IEC 60730-1 2013 which includes the "*Controls for Software*" detailed in Annex H.11.12.

Aside from the blanket "*compliance check*" previously mentioned, subclauses H.11.12.1 to H.11.12.4 inclusive are only applicable to control functions using software class B or class C, and include measures for the avoidance of systemic faults. Subclause H.11.12.4 contains additional requirements for remotely actuated control functions.

The V model in Figure 1 is extracted from IEC 61508-3 and adapted to the needs of IEC 60730.

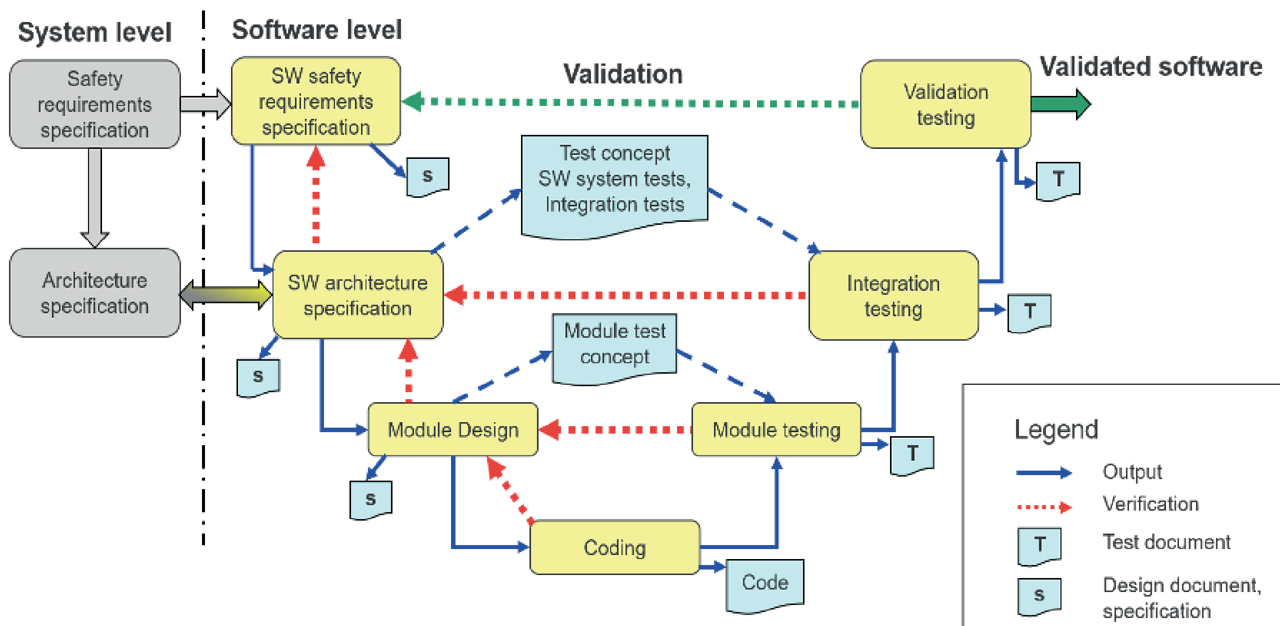

Figure 1: IEC 60730 V-model is adapted from that of IEC 61508-3

[13] Controller for a washing machine (IEC 60730 / IEC 60335, 'Class B')
https://www.safetty.net/tt-design-examples/iec-60730-washer
[14] IEC 60335-1:2010 Household and similar electrical appliances – Safety
https://webstore.iec.ch/publication/1499
[15] IEC 60730-1:2013+AMD1:2015+AMD2:2020 CSV Consolidated version Automatic electrical controls
https://webstore.iec.ch/publication/66894
[16] IEC 60730-1:2013+AMD1:2015+AMD2:2020 §H.11.12 "Controls using software"
[17] IEC 60730-1:2013+AMD1:2015+AMD2:2020 §H.2.22 "Definitions relating to classes of control functions"

LDRA

Requirements for the architecture (Clause H.11.12.1)

Designers of control functions deploying software of class B or C are required to select from one of a number of architectural strategies defined in the standard. Each approach is designed to control and avoid software-related faults and errors in safety-related data and safety-related segments of the software. These are listed in Figure 2.

Class	Control functions with software	Description
B	Single channel with functional test	A single channel structure in which test data is introduced to the functional unit prior to its operation
B	Single channel with periodic self-test	A single channel structure in which components of the control are periodically tested during operation
B	Dual channel without comparison	A dual channel structure which contains two mutually independent functional means to execute specified operations
C	Single channel with periodic self-test and monitoring	A single channel structure in which components of the control are periodically tested during operation, and monitored on an ongoing basis
C	Dual channel (homogenous) with comparison	A dual channel structure containing two identical and mutually independent functional means, each capable of providing a declared response, in which comparison of internal signals or output signals is performed for fault/error recognition
C	Dual channel (diverse) with comparison	A dual channel structure containing two different and mutually independent functional means, each capable of providing a declared response, in which comparison of output signals is performed for fault/error recognition

Figure 2: Structure of control software classes

Measures to control faults/errors (Clause H.11.12.2)

Software diversity, a form of dual redundancy, is a principle favoured by the standard for the control of faults or errors in the software. For example, "*redundant memory with comparison*" requires the use of different data formats to record the same data on two areas of the same component. Additional fault detection means such as periodic functional test, periodic self-tests, independent monitoring are required for the detection of faults that are not covered by comparison.

It is also recommended to provide means for the recognition and control of errors in transmissions to external safety-related data paths. For classes B and C, it is required that measures should be implemented to address faults or errors in safety-related segments and data. Figure 3 shows some examples of fault control techniques applicable to peripherals.

Component in MCU	Fault/Error	Class B	Class C	Example Measure
Clock	Wrong Frequency	Recommended	Recommended	Frequency monitoring by reciprocal comparison independent hardware comparator
Variable memory	DC fault or dynamic cross-links	Recommended	Recommended	Periodic static memory test or word protection with single bit redundancy, redundant memory with comparison

Figure 3: Examples of peripheral fault control techniques

Measures to avoid errors (Clause H.11.12.3)

Systemic failure can be defined as "*A failure that happens in a deterministic (non- random) predictable fashion from a certain cause, which can only be eliminated by a modification of the design or of the manufacturing process, operational procedures, documentation, or other relevant factors.*"[18]

Failures resulting from software problems are almost always systemic in nature, and the safety lifecycle activities illustrated in Figure 4 are designed to avoid them. The verification of adherence to the recommended practices applicable to each of the lifecycle stages is required to qualify the software for use in Class B and Class C appliances. The following subsections will explore those practices further.

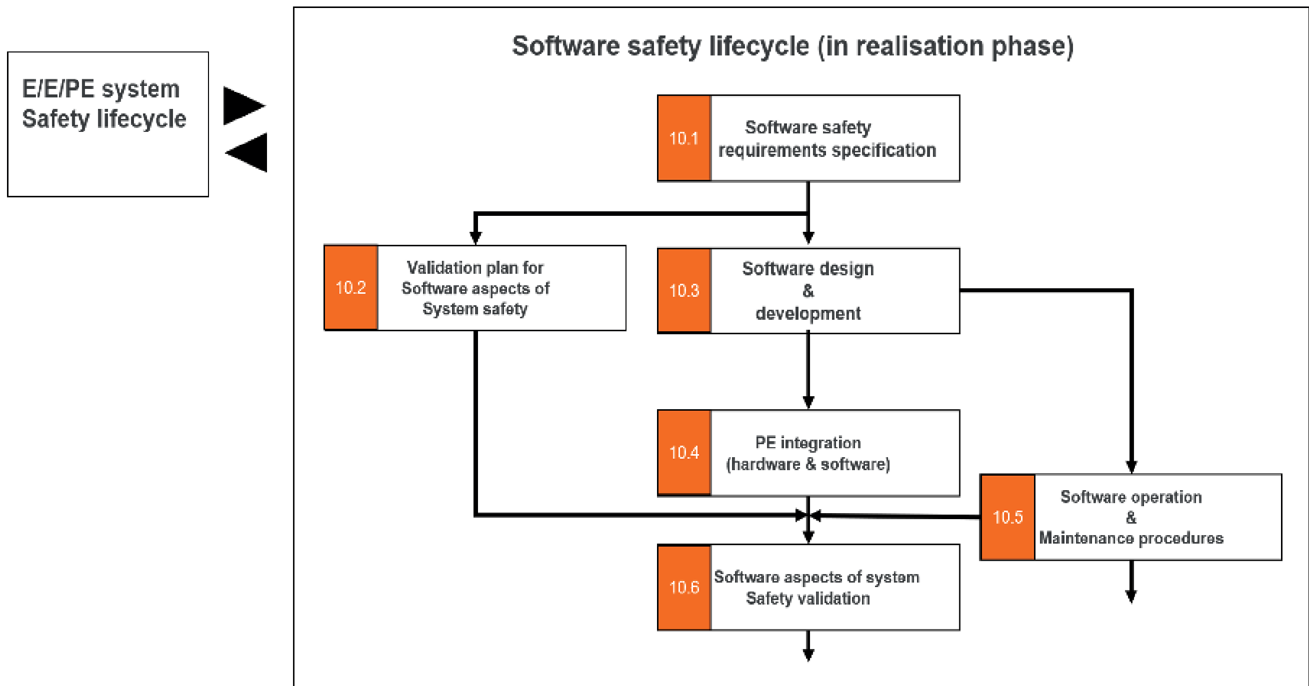

Figure 4: The Software Safety Lifecycle

Specification (Clause H.11.12.3.2)

Software safety requirements (Clause H.11.12.3.2.1)

During the control system design phase, functional requirements and safety requirements are refined, and software and hardware elements are identified.

The primary objective of specification for the resulting software safety requirements is to describe every safety-related function and non-safety-related function to be implemented, including functions related to the detection, annunciation, and management of software and hardware faults. These descriptions should include details of response times, related software classes, and interfaces between hardware and software.

The secondary objective of this phase is to review and update the safety-related requirements previously identified in the context of the system as a whole, referencing hardware and software interfaces, data flow, data storage, data processing, and any subsystems supporting safety functionality.

As part of the standard's requirements verification activities (which also include the verification of system-level requirements coverage, for example) the review will consider whether the requirements have been defined in accordance with best practice characteristics and attributes for good requirements are followed. Establishing traceability for backward and forward requirements coverage ensures that all requirements are met.

[18] Exida resources – Systemic failure
https://www.exida.com/Resources/Term/systematic_failure

Techniques and measures can be applied in accordance with IEC 61508 as shown in Figure 5, with Figure 6 showing how that principle applies in the case of software safety requirements specification.

Standard	Classification			
IEC 61508	SIL 1	SIL 2	SIL 3	SIL 4
IEC 60730	Class A	Class B		Class C

Figure 5: Mapping of IEC 60730 classes to IEC 61508 SILs

Although the standard does not require the use of tools, they can help make compliance far more efficient. Requirement management tools are often used to specify and manage the requirements. Verification and validation tools are used to create artefacts demonstrating that the products of development are in accordance with the standard. And requirements traceability tools are used to demonstrate that requirements are completely and uniquely covered by the resulting system.

Technique/Measure Software safety requirements specification		**Ref**	**SIL**			
			1	**2**	**3**	**4**
			Class			
			A	**B**		**C**
1a	Semi-formal methods	Table B.7	**R**	**R**	**HR**	**HR**
1b	Formal methods	B.2.2, C.2.4	**---**	**R**	**R**	**HR**
2	Forward traceability between the system safety requirements and the software safety requirements	C.2.11	**R**	**R**	**HR**	**HR**
3	Backward traceability between the safety requirements and the perceived safety needs	C.2.11	**R**	**R**	**HR**	**HR**
4	Computer-aided specification tools to support appropriate techniques/measures above	B.2.4	**R**	**R**	**HR**	**HR**
"HR" The method is highly recommended for this class. **"R"** The method is recommended for this class. **"---"** The method has no recommendation for or against its usage for this class.						

Figure 6: Copy of IEC 61508-3 Table A.1[19] mapped to IEC 60730. LDRA static analysis tools support the highlighted techniques.

Figure 7 shows a requirement coverage report generated automatically from the LDRA tool suite, linking system level requirements to software requirements. An interface between the requirements management tool of choice and the LDRA tool suite provides access to the requirements, and allows the percentage of coverage for forward and backward traceability to be calculated.

The traceability matrix report shown represents requirements coverage in an intuitive way, allowing any gaps to be easily identified. The reports can also be considered to be verification artefacts in accordance with IEC 60730.

[19] IEC 61508-3 Annex A Table A.1 – Software safety requirements specification

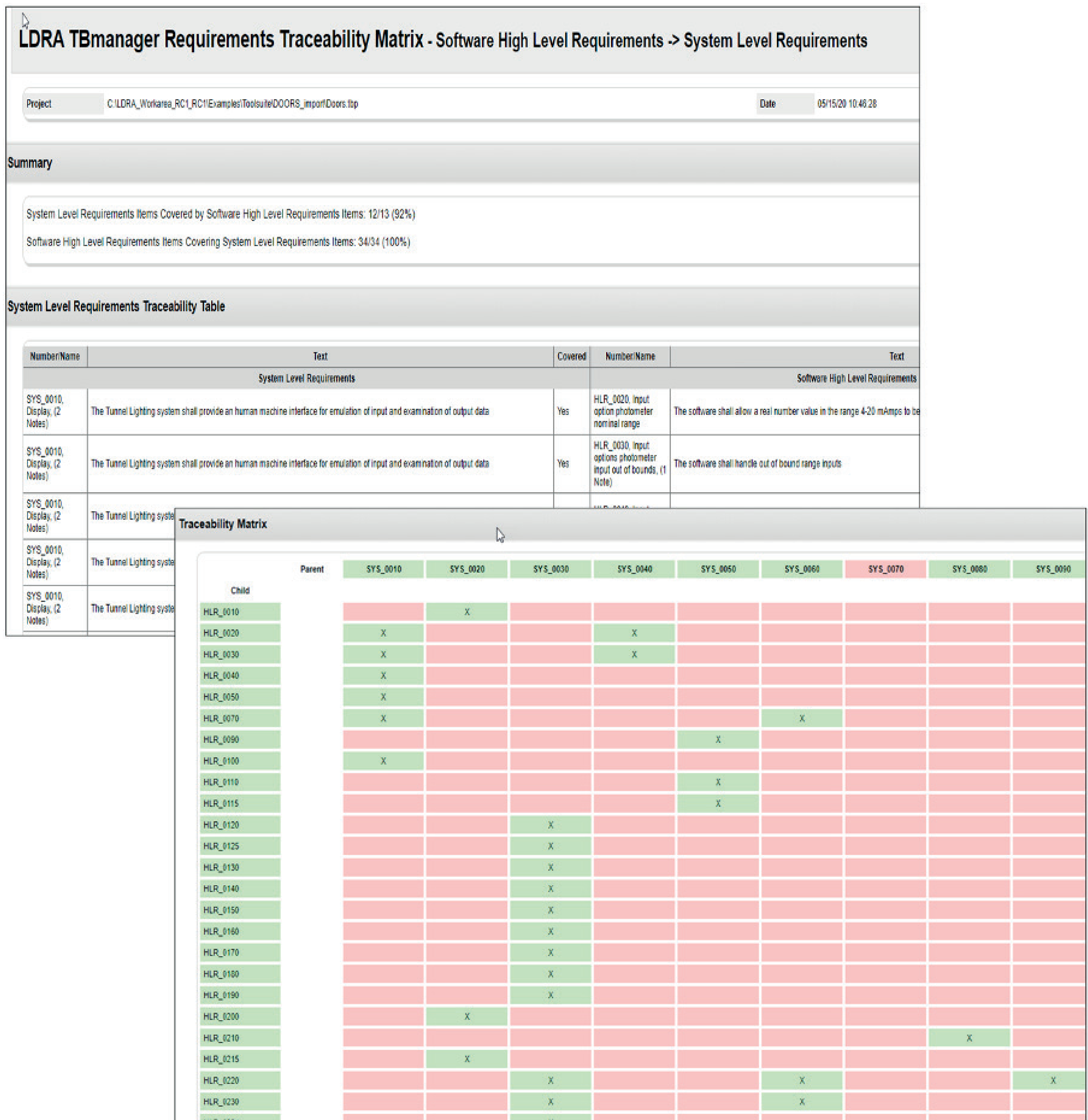

Figure 7: Requirement coverage reporting in the LDRA tool suite

Software architecture (Clause H.11.12.3.2.2)

The primary objective of this clause is to ensure that the specified software architecture fulfils the standard's requirements for the relevant control class.

The architecture is required to be analysable and verifiable, and capable of being modified without compromising safety. The design specification techniques are detailed in Table A.2 of IEC 61508 with respect to the static and dynamic design aspects. During the planning phase of the software development activities, techniques are nominated from that table as appropriate to the application and its classification.

Figure 8 and Figure 9 list the techniques and provides a cross reference to those architecture and design features that can be confirmed by the LDRA tool suite as being reflected in the resulting source code.

	Technique/Measure Architecture and design features	Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B	Class B	Class C
1a	Fault detection	C.3.1	R	R	HR	HR
2	Error detecting codes	C.3.2	---	R	R	HR
3a	Failure assertion programming	C.3.3	R	R	HR	HR
3b	Diverse monitor techniques (with independence between the monitor and the monitored function in the same computer)	C.3.4	R	R	HR	HR
3c	Diverse monitor techniques (with separation between the monitor computer and the monitored computer)	C.3.4	R	R	HR	HR
3d	Diverse redundancy, implementing the same software safety requirements specification	C.3.5	---	---	---	R
3e	Functionally diverse redundancy, implementing different software safety requirements specification	C.3.5	---	---	R	HR
3f	Backward recovery	C.3.6	R	R	---	HR
3g	Stateless software design (or limited state design)	C.2.12	---	---	R	HR
4a	Re-try fault recovery mechanisms	C.3.7	R	R	---	---
4b	Graceful degradation	C.3.8	R	R	HR	HR
5	Artificial intelligence - fault correction	C.3.9	---	NR	NR	NR
6	Dynamic reconfiguration	C.3.10	---	NR	NR	NR
7	Modular approach	Table B.9	HR	HR	HR	HR
8	Use of trusted/verified software elements (if available)	C.2.10	R	HR	HR	HR

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

Figure 8 (Part 1): Copy of IEC 61508-3 Table A.2[20] as referenced by IEC 60730. The architecture and design features selected for use should subsequently be reflected in both the design and the resulting source code. Highlighted features can be verified by the LDRA tool suite as being implemented in that code.

[20] IEC 61508-3 Annex A Table A.2 – Software design and development – software architectural design

	Technique/Measure Architecture and design features	Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B	Class B	Class C
9	Forward traceability between the software safety requirements specification and software architecture	C.2.11	R	R	HR	HR
10	Backward traceability between the software safety requirements specification and software architecture	C.2.11	R	R	HR	HR
11a	Structured diagrammatic methods **	C.2.1	HR	HR	HR	HR
11b	Semi-formal methods • Logical/functional block diagrams • Sequence diagrams • Finite state machines/state transition diagrams Dataflow diagram	Table B.7	R	R	HR	HR
11c	Formal design and refinement methods **	B2.2, C2.4	---	R	R	HR
11d	Automatic software generation	C.4.6	R	R	R	R
12	Computer-aided specification and design tools	C.2.4	R	R	HR	HR
13a	Cyclic behaviour, with guaranteed maximum cycle time	C.3.11	R	HR	HR	HR
13b	Time-triggered architecture	C.3.11	R	HR	HR	HR
13c	Event-driven, with guaranteed maximum response time	C.3.11	R	HR	HR	---
14	Static resource allocation	C.2.6.3	---	R	HR	HR
15	Static synchronization of access to shared resources	C.2.6.3	---	---	R	HR

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

Figure 9 (Part 2): Copy of IEC 61508-3 Table A.2 as referenced by IEC 60730. The architecture and design features selected for use should subsequently be reflected in both the design and the resulting source code. Highlighted features can be verified by the LDRA tool suite as being implemented in that code.

A secondary objective of this subclause is to ensure that the software is designed and implemented in accordance with the techniques and measures appropriate to its nominated class. Verification plays a critical role and the requirements for safety-related software need to be verified at design level using established methods such as control flow analysis, data flow analysis, walk-throughs, and design reviews.

The architectural specification is to be verified as being in accordance with the specification of the software safety requirements to ensure the correctness of:

- interactions between hardware and software,
- partitioning into modules and their allocation to the specified safety functions,
- hierarchy and call structure of the modules (control flow) (Figure 10),
- data flow and restrictions on data access (Figure 10), and
- architecture and storage of data.

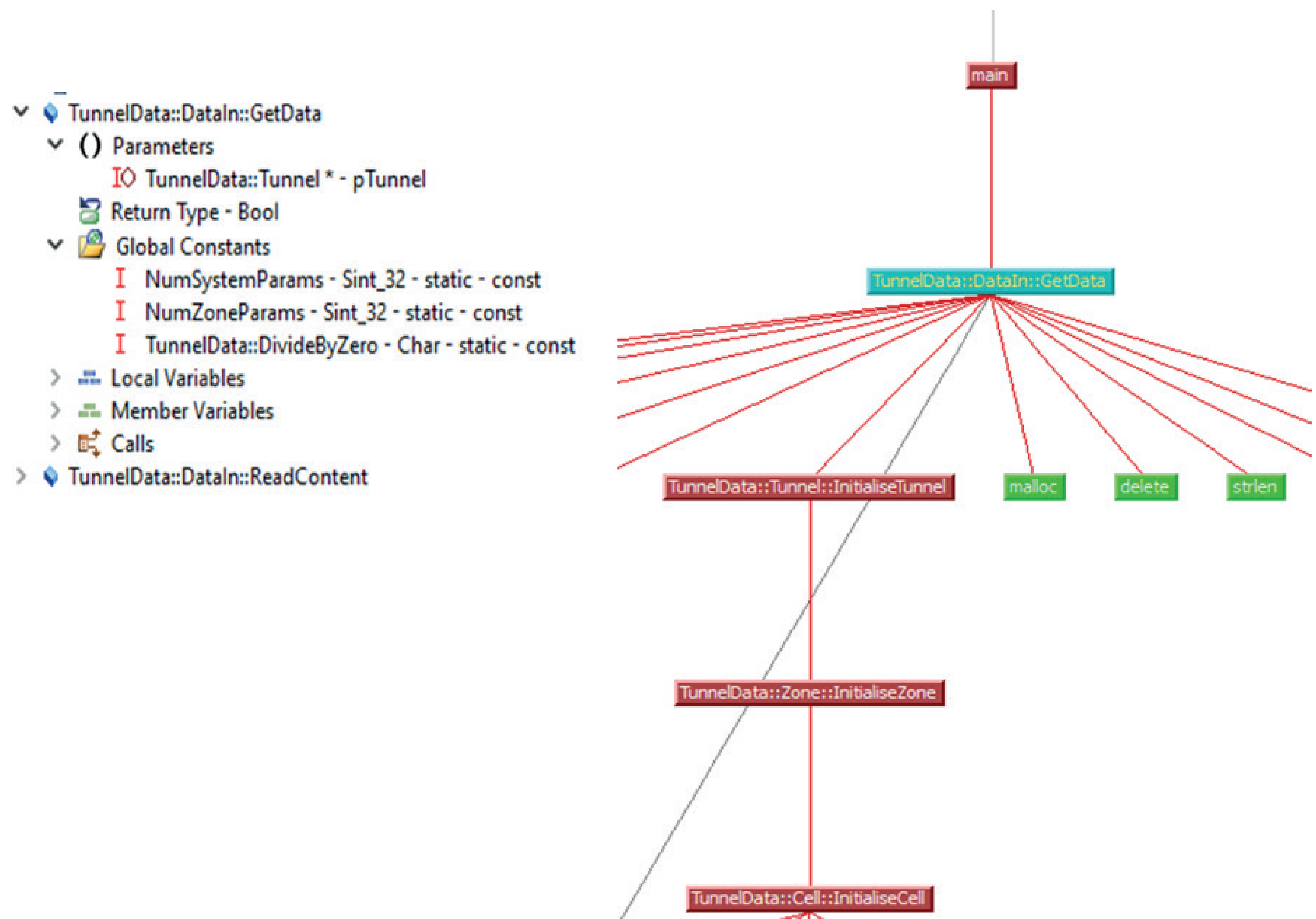

Figure 10: Diagrammatic representations of data flow (left) and control flow generated from source code by the LDRA tool suite aid verification of the implementation of software architectural design.

Figure 11 illustrates the standard's guidance with regards to the selection of the programming language(s) to be used and the associated tool chain for the development of that code, including verification and validation tools, static code analysers, test coverage monitors and configuration management tools.

Technique/Measure Support tools and programming languages		Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B	Class B	Class C
1a	Suitable programming language	C.4.5	HR	HR	HR	HR
1b	Strongly typed programming language	C.4.1	HR	HR	HR	HR
2	Language subset	C.4.2	---	---	HR	HR
3	Certified tools and certified translators	C.4.3	R	HR	HR	HR
4	Tools and translators: increased confidence from use	B.4.4	HR	HR	HR	HR
"HR" The method is highly recommended for this class. **"R"** The method is recommended for this class. **"---"** The method has no recommendation for or against its usage for this class.						

Figure 11: Copy of IEC 61508-3 Table A.3[21] as referenced by IEC 60730. The techniques selected for use should subsequently be reflected in both the design and the resulting source code. Highlighted features can be verified by the LDRA tool suite as being implemented in that code.

Module design and coding (Clause H.11.12.3.2.3)

Software is required to be designed in accordance with modular principles, and to reflect the architectural design, such that the design and the resulting code is traceable to the software architecture, and hence to requirements. The design is required to specify function(s), interfaces to other modules, and data.

The best practise design principles of maintaining the hierarchical structure with minimized data and control flow can be achieved in this phase. Structural complexity can be minimized by keeping the number of possible paths through each software module small, and the relationship between the input and output parameters can be kept as simple as possible by avoiding complicated branching and any unconditional jumps in higher level languages.

Defensive programming and plausibility checks can also be adopted within the modules. The recommended techniques and measures can be found in Figure 12, which also illustrates how the LDRA tool suite can help.

[21] IEC 61508-3 Annex A Table A.2 – Software design and development – support tools and programming languages

Technique/Measure Detailed design		Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B	Class B	Class C
1a	Structured methods **	C.2.1	HR	HR	HR	HR
1b	Semi-formal methods **	Table B.7	R	HR	HR	HR
1c	Formal design and refinement methods **	B.2.2 C.2.4	---	R	R	HR
2	Computer-aided design tools	B.3.5	R	R	HR	HR
3	Defensive programming	C.2.5	---	R	HR	HR
4	Modular approach	Table B.9	HR	HR	HR	HR
5	Design and coding standards	C.2.6 Table B.1	R	HR	HR	HR
6	Structured programming	C.2.7	HR	HR	HR	HR
7	Use of trusted/verified software elements (if available)	C.2.10	R	HR	HR	HR
8	Forward traceability between the software safety requirements specification and software design	C.2.11	R	R	HR	HR

“HR” The method is highly recommended for this class.
“R” The method is recommended for this class.
“---” The method has no recommendation for or against its usage for this class.
** Group 1, , “Structured methods”. Use measure 1a only if 1b is not suited to the domain for SIL 3R4.

Figure 12: Copy of IEC 61508-3 Table A.4[22] as referenced by IEC 60730. The techniques selected for use should subsequently be reflected in the resulting source code. Highlighted features can be verified by the LDRA tool suite as being implemented in that code.

Static analysis techniques including control flow analysis, data flow analysis, walk-throughs, and design reviews can be applied in order to confirm that the module specification is in accordance with the architecture specification.

Model based development

The LDRA tool suite can be integrated with several different model-based development tools exemplified by IBM Engineering Systems Design Rhapsody[23], MathWorks Simulink[24] and Ansys SCADE[25]. 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.

[22] IEC 61508-3 Annex A Table A.4 – Software design and development – Detailed design
[23] IBM Engineering Systems Design Rhapsody https://www.ibm.com/us-en/marketplace/systems-design-rhapsody
[24] MathWorks Simulink - Simulation and Model-Based Design https://www.mathworks.com/products/simulink.html
[25] Ansys Scade https://www.ansys.com/products/embedded-software

Design and coding standards (Clause H.11.12.3.2.4)

This clause describes the phase in which code is designed and developed, applying the design practices and coding standards specified earlier in the lifecycle. Coding standards look to define programming practice including naming conventions, proscribe unsafe language features, and specify procedures for source code documentation. Static analysis techniques are used to verify that the resulting application code represents and accurate interpretation of the module specification.

By applying these best practices, the resulting code will be as secure, reliable, error-free, and easy to test and maintain as possible. For example:

- Large, rambling functions with complex interfaces are difficult to read, maintain, and test – and hence more susceptible to error.
- High cohesion improves maintainability and reduces complexity. (sidebar)

These measures prescribed by the standard can be checked quickly using automated tools, such as the TBvision component of the LDRA tool suite (Figure 13). TBvision can be used to evaluate the use of interrupts, pointers, recursion, and non-structured control flow, to check for run time errors, and to perform Structured Programming Verification (SPV) to ensure that there are no potentially harmful unstructured sections in the application code.

The term "cohesion" refers to the "degree to which the elements inside a module belong together".

Advantages of high cohesion include:

- Reduced module complexity
- Increased system maintainability, because logical changes in the do main affect fewer modules.

	Technique/Measure Design and coding standards	Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B	Class B	Class C
1	Use of coding standard to reduce likelihood of errors	C.2.6.2	HR	HR	HR	HR
2	No dynamic objects	C.2.6.3	R	HR	HR	HR
3a	No dynamic variables	C.2.6.3	---	R	HR	HR
3b	Online checking of the installation of dynamic variables	C.2.6.4	---	R	HR	HR
4	Limited use of interrupts	C.2.6.5	R	R	HR	HR
5	Limited use of pointers	C.2.6.6	---	R	HR	HR
6	Limited use of recursion	C.2.6.7	---	R	HR	HR
7	No unstructured control flow in programs in higher level languages	C.2.6.2	R	HR	HR	HR
8	No automatic type conversion	C.2.6.2	R	HR	HR	HR

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

Figure 13: Copy of IEC 61508-3 Table B.1[26] as referenced by IEC 60730. Highlighted techniques and measures are supported by the LDRA tool suite.

[26] IEC 61508-3 Annex B Table B.1 – Design and coding standards

Verification of implemented modules

Best practise dictates that static and dynamic analysis of the code should be an ongoing process while ever it is under development. The code implementation process is therefore interwoven with ongoing static analysis, and with module and integration testing.

The are many internationally recognised coding standards, including MISRA C:2012, MISRA C++:2008, JSF++ AV, and CERT. Both IEC 61508 and IEC 60730 make it clear that development teams are at liberty to adapt any these standards or even to develop their own. The adherence of application code to the standard of choice can be verified within the LDRA tool suite to ensure that any detrimental effect on productivity resulting from the adherence to coding standards is kept to a minimum (Figure 14).

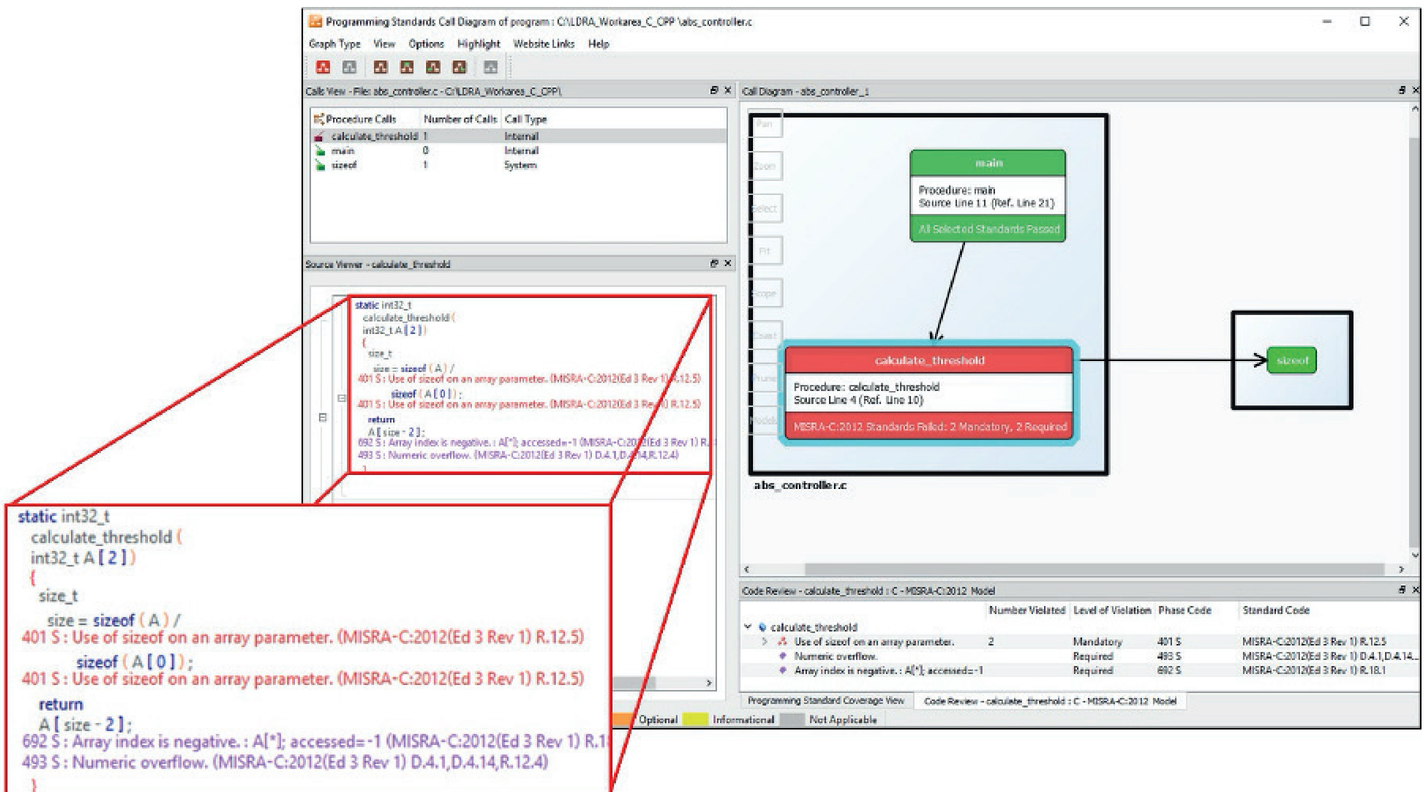

Figure 14: Highlighting coding guideline violations according to MISRA C:2012 Ed3, Rev1.

In practice, the ongoing application of static analysis throughout the code implementation phase can provide support and tutelage to a development team. For developers who are newcomers to IEC 60730, the role of the tool often evolves from a means to highlight where violations have occurred, to one where it provides confirmation that there are none.

Figure 15 shows how the standard calls for the application of several review processes and analysis with the aim of producing clear, maintainable and testable code. The TBvision component of the LDRA tool suite includes several features to help achieve these aims, including the generation of software quality metrics.

	Technique/Measure Static analysis	Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B		Class C
1	Boundary value analysis	C.5.4	R	R	HR	HR
2	Checklists	B.2.5	R	R	R	R
3	Control flow analysis	C.5.9	R	HR	HR	HR
4	Data flow analysis	C.5.10	R	HR	HR	HR
5	Error guessing	C.5.5	R	R	R	R
6a	Formal inspections, including specific criteria	C.5.14	R	R	HR	HR
6b	Walk-through (software)	C.5.15	R	R	R	R
7	Symbolic execution	C.5.11	---	---	R	R
8	Design review	C.5.16	HR	HR	HR	HR
9	Static analysis of run time error behaviour	B.2.2 C.2.4	R	R	R	HR
10	Worst-case execution time analysis	C.5.20	R	R	R	R

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

Figure 15: Copy of IEC 61508-3 Table B.8[27] as referenced by IEC 60730. Highlighted techniques and measures are supported by the LDRA tool suite.

These metrics provide a means to ensure that software component size, complexity, cohesion, and coupling are controlled. Complexity metrics, for example, are generated through a combination of interface analysis, cohesion evaluated through data object analysis, and coupling through data control coupling analysis.

Testing (Clause H.11.12.3.3)

Software test is performed across a number of stages as development progress.

Module level testing is first to ensure that modules have been implemented in accordance with the low-level design specification and hence fulfil all specified safety functions and control functions. Unintended functionality must be also be shown to be absent.

As software modules are integrated together, testing of the resulting software subassemblies and ultimately the complete integrated system are validated with suitable test cases based on the software safety requirements specification.

In general, the use of a fully integrated tool suite for testing can ensure that the good practices required by IEC 60730 are adhered to whether they are coding rules, design principles, or principles for software architectural design.

[27] IEC 61508-3 Annex B Table B.8 – Static analysis

Module design testing (Clause H.11.12.3.3.1)

The objective of the code reviews during the module design and implementation phase is to incorporate good coding practices and ensure that the implemented software is of high quality (Figure 16).

A test concept with suitable test cases is required, based on the low level module design specification. Each software module is then tested as specified within that test concept with test cases, data and results documented. Code verification of a software module by static means includes such techniques as software inspections, walk-throughs, static analysis and formal proofs.

Code verification of a software module by dynamic means includes functional testing, white box testing and statistical testing. Where model-based development is deployed, back-to-back testing at the model and code level is recommended.

It is the combination of evidence collated from both dynamic and static analysis that provides assurance that each software module satisfies its associated specification. Software unit and integration tests need to be executed on target hardware and if the developed unit or integrated software is "safety-related", then test results should comply with safety requirements.

Fault injection and resource tests help further ensure robustness and resilience.

	Technique/Measure Software module testing and integration	**Ref**	**SIL**			
			1	**2**	**3**	**4**
			Class			
			A	**B**		**C**
1	Probabilistic testing	C.5.1	---	**R**	**R**	**R**
2	Dynamic analysis and testing	B.6.5 Table B.2	**R**	**HR**	**HR**	**HR**
3	Data recording and analysis	C.5.2	**HR**	**HR**	**HR**	**HR**
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	**HR**	**HR**	**HR**	**HR**
5	Performance testing	Table B.6	**R**	**R**	**HR**	**HR**
6	Model based testing	C.5.27	**R**	**R**	**HR**	**HR**
7	Interface testing	C.5.3	**R**	**R**	**HR**	**HR**
8	Test management and automation tools	C.4.7	**R**	**HR**	**HR**	**HR**
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	**R**	**R**	**HR**	**HR**
10	Formal verification	C.5.12	---	---	**R**	**R**

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

Figure 16: Copy of IEC 61508-3 Table A.5[28] as referenced by IEC 60730. Highlighted techniques and measures are supported by the LDRA tool suite.

[28] IEC 61508-3 Annex A Table A.5 – Software design and development –Software module testing and integration

Although module testing can be performed by writing custom code for the purpose, the use of a certified, proven test tool is likely to be much more cost effective unless the code base is very small. Such a tool can automatically generate test drivers and harnesses (wrapper code) with no extra coding or scripting required, enabling tests to be easily and efficiently run on code units. These tests can be subsequently regressed, with clear maintenance tracking and seamless storage of test data and results. An illustration of requirements-based unit testing using the TBrun component of the LDRA tool suite is shown in Figure 17.

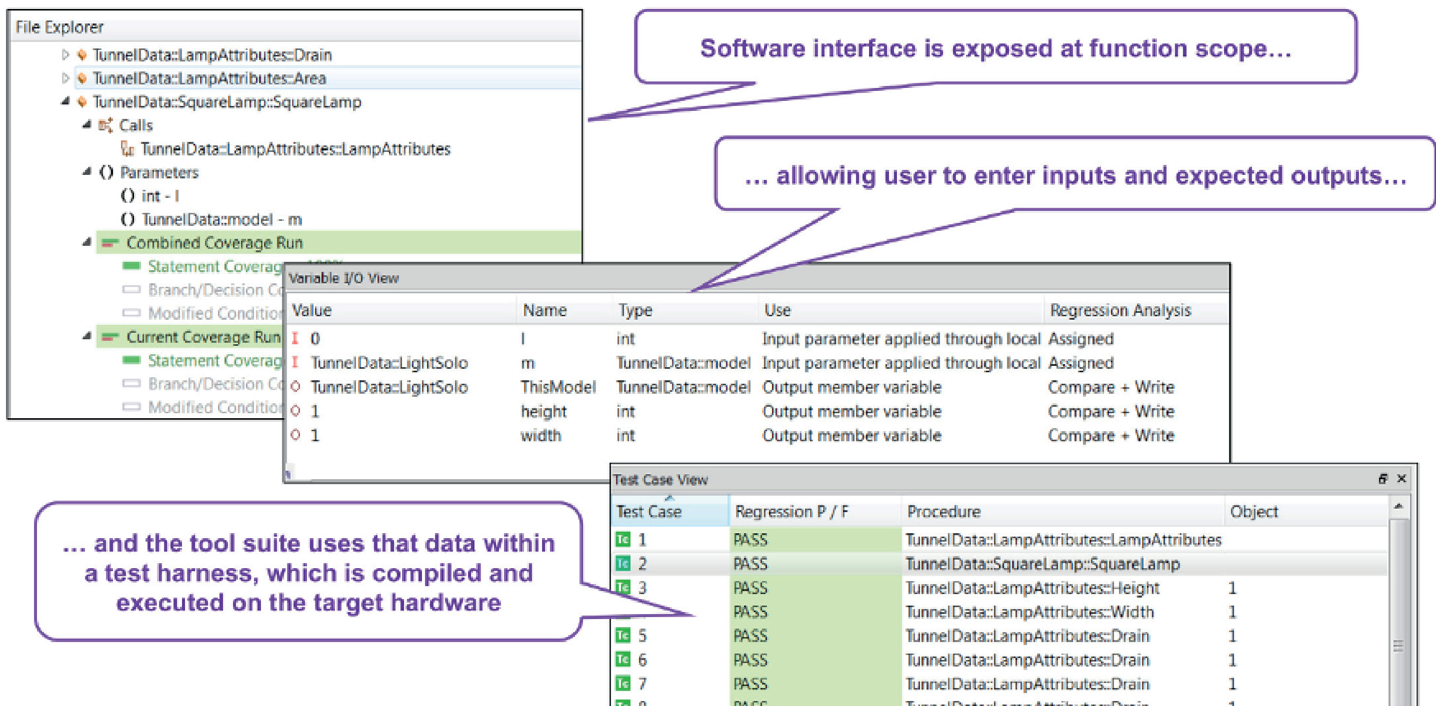

Figure 17: Requirements-based unit testing using the TBrun component of the LDRA tool suite

Structural coverage metrics

In addition to showing that the software functions correctly, dynamic analysis is also used to generate structural coverage metrics. In tandem with the coverage of requirements at the software unit level, these metrics provide the necessary data to evaluate the completeness of test cases and to demonstrate that there is no unintended functionality. Statement, branch and MC/DC coverage are provided by both the unit test and system test facilities of the LDRA tool suite. Various test methods are applied during unit and integration testing as listed in Figure 18.

	Technique/Measure Dynamic analysis and testing	Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class A	Class B		Class C
1	Test case execution from boundary value analysis	C.5.4	R	HR	HR	HR
2	Test case execution from error guessing	B.5.5	R	R	R	R
3	Test case execution from error seeding	C.5.6	---	R	R	R
4	Test case execution from model-based test case generation	B.5.27	R	R	HR	HR
5	Performance modelling	C.5.20	R	R	R	HR
6	Equivalence classes and input partition testing	C.5.7	R	R	R	HR
7a	IStructural test coverage (entry points) 100 % **	C.5.8	HR	HR	HR	HR
7b	Structural test coverage (state-ments) 100 %**	C.5.8	R	HR	HR	HR
7c	Structural test coverage (branch-es) 100 %**	C.5.8	R	R	HR	HR
7d	Structural test coverage (condi-tions, MC/DC) 100 %**	C.5.8	R	R	R	HR

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

Figure 18: Copy of IEC 61508-3 Table B.2[29] as referenced by IEC 60730. Highlighted techniques and measures are supported by the LDRA tool suite.

Figure 19 shows how structural coverage can be seen graphically in the control flow graphs and html reports of the LDRA tool suite.

[29] IEC 61508-3 Annex A Table B.2 – Dynamic analysis and testing

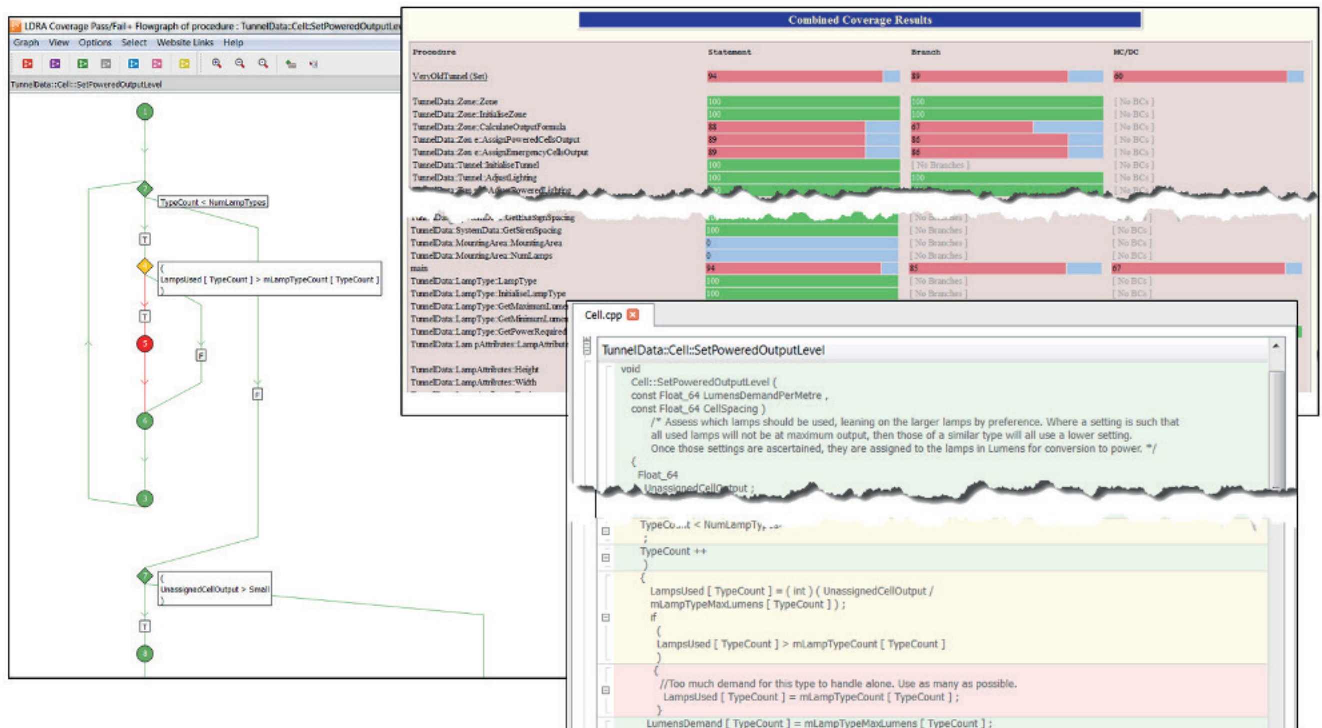

Figure 19: Structural coverage can be seen graphically with control flow graphs and html reports using the LDRA tool suite

Software integration testing (Clause H.11.12.3.3.2)

This clause requires that a test concept with suitable test cases is to be defined based on the architecture design specification, and that the software is to be tested as specified within that test concept. Test cases, test data and test results are to be documented.

Integrated software is to be proven by means of a number of specified test techniques. Depending on the class of software, these may include functional and "*black box*" tests used to check the dynamic behaviour of the software under realistic functional conditions, with the aim of revealing any failures to meet the functional specification.

Test data may include combinations of:

- permissible ranges,
- inadmissible ranges,
- range limits, and
- extreme values.

Testing is to be the main validation method for software, and modelling can be used to supplement the validation activities (Figure 20).

Integration testing is designed to ensure that when the units are working together in accordance with the software architectural design, they meet the related specified requirements. In practice, these integration tests typically involve the verification of safety and non-safety related software functions.

In general, it is desirable for all dynamic testing to use environments which correspond closely to the target environment and hence test dependencies between hardware and software. However, that is not always practical and one approach involves developing the tests in a simulated environment and then, once proven, re-running them on the target.

	Technique/Measure Programmable electronics integration (hardware and software)	Ref	SIL 1	SIL 2	SIL 3	SIL 4
			Class			
			A	B		C
1	Functional and black box testing – Boundary value analysis – Process simulation	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
2	Performance testing – Finite state machines	Table B.6	R	R	HR	HR
3	Forward traceability between the system and software design requirements for hardware software integration and the hardware/software integration test specifications	C.2.11	R	R	HR	HR

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

Figure 20: Copy of IEC 61508-3 Table A.6[30] as referenced by IEC 60730. Highlighted techniques and measures are supported by the LDRA tool suite.

To complement the structural coverage analysis (discussed in relation to module level testing), robustness tests including boundary values could be provided manually or generated automatically (Figure 21) to verify system behaviour in response to both permissible and inadmissible data ranges.

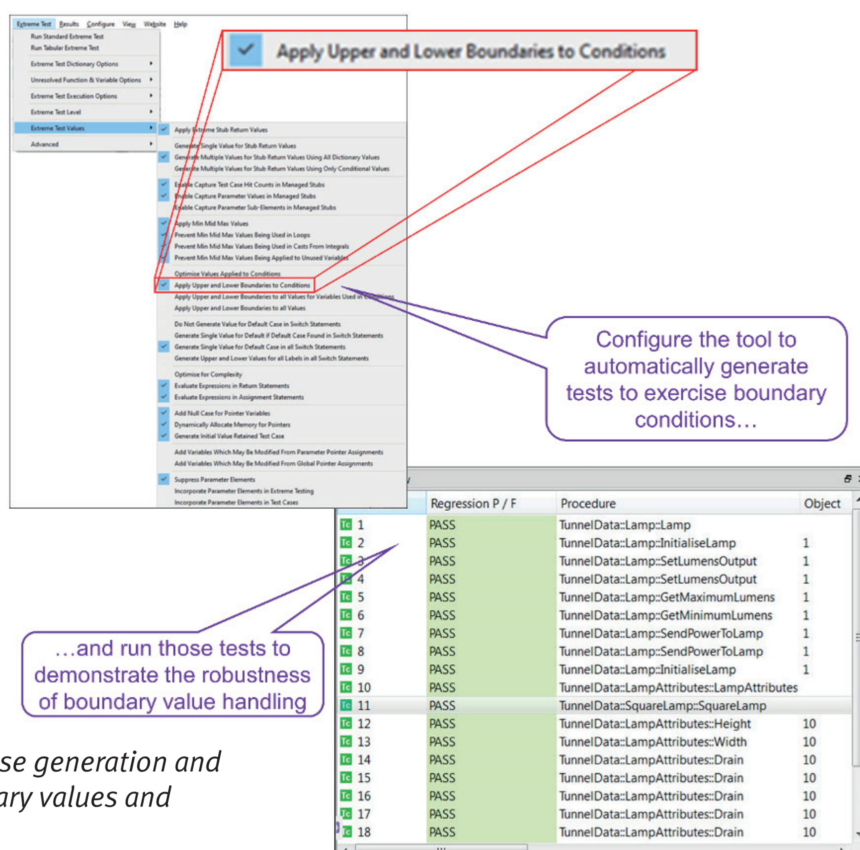

Figure 21: Automatic test case generation and input population for boundary values and robustness test cases

[30] IEC 61508-3 Annex A Table A.6 - Programmable electronics integration (hardware and software)

Traceability

Establishment of forward and backward traceability is one of the requirements during module testing and integration testing to ensure all requirements have been covered and all implementation has been tested adequately. Tracing the low level requirements to source code and test cases can challenging, because of the different tools typically used for requirement management and source code development.

The TBmanager component of the LDRA tool suite can help to establish traceability horizontally and vertically throughout the lifecycle to source code, requirements and test artefacts.

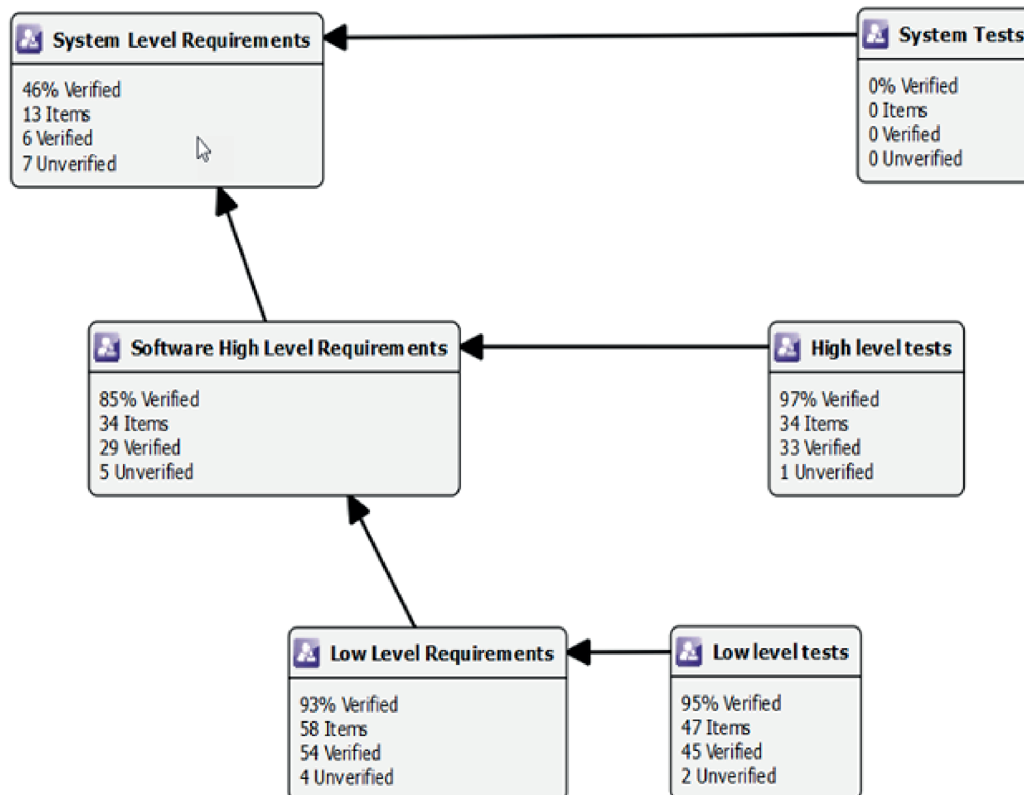

Figure 22: Performing requirements based testing. Test cases are linked to requirements and executed within the LDRA tool suite.

Figure 22 shows the traceability establishment for the lifecycle stages using the TBmanager component of the LDRA tool suite.

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 the IBM Engineering Requirements DOORS Family[31], Siemens PLM Polarion ALM[32], or any ALM tool supporting the standard Requirements Interchange Format[33].

TBmanager integrates with these requirements management tools, mapping requirements to source code implementation at module or integration level. It shows the fulfilment of low-level requirements, high-level requirements, and/or the architectural specification, and creates an association with the artefacts created by tools at all stages in the lifecycle (Figure 23).

[31] IBM Engineering Requirements Management DOORS Family
https://www.ibm.com/us-en/marketplace/requirements-management

[32] Siemens - Software Lifecycle Under Control
https://polarion.plm.automation.siemens.com/

[33] Object Management Group – Requirements Interchange Format
http://www.omg.org/spec/ReqIF/

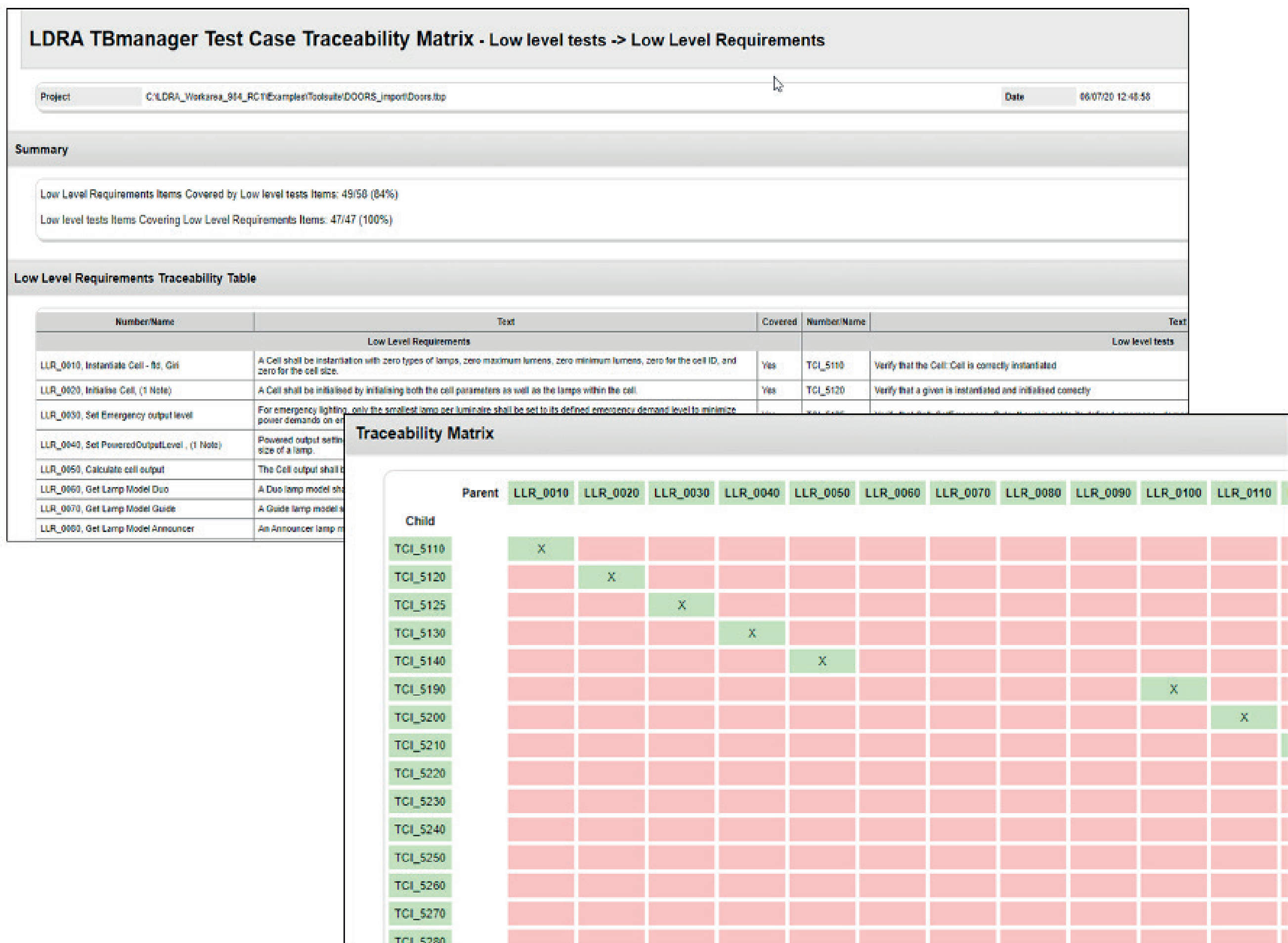

Figure 23: Reporting in the TBmanager component of the LDRA tool suite, providing traceability between low level requirements and test cases.

Software validation (Clause H.11.12.3.3.3)

This section of IEC 60730 deals with the software aspects of system safety validation, ensuring that the integrated system complies with the software safety requirements specification in accordance with the specified class.

A validation concept with suitable test cases is created based on the software safety requirements specification which is then used to validate the software. The software is exercised by simulation or stimulation of:

- input signals present during normal operation,
- anticipated occurrences,
- undesired conditions requiring system action.

Test cases, test data and test results are documented.

The techniques and measures deployed are similar to those applied during integration, as shown in Figure 20.

Functional and black box testing can be used to check whether the functions of a system or program behave as the specification dictates when executed in a prescribed environment according to established criteria. The associated configuration files can be stored and used for the automated regression analysis to confirm ongoing adherence to the specified requirements.

Automated requirements traceability tools like the LDRA tool suite complement this concept by providing forward and backward traceability between the software safety requirements specification and software safety validation plan.

Tools, programming languages, management of software versions, and modification (Clause H.11.12.3.4)

Equipment used for software design, verification and maintenance, such as design tools, programming languages, translators and test tools, need to be qualified appropriately and shown to be fit for purpose. IEC 60730 states that the tools are assumed to be suitable if "increased confidence from use" can be demonstrated in accordance with C.4.4 of IEC 61508-7:2010. Figure 24 shows the techniques or measures for support tools and programming language during software design and development.

	Technique/Measure Support tools and programming language	**Ref**	**SIL**			
			1	**2**	**3**	**4**
			Class			
			A	**B**		**C**
1	Suitable programming language	Table A.3	**HR**	**HR**	**HR**	**HR**
2	Strongly typed programming language	Table A.3	**R**	**R**	**HR**	**HR**
3	Language subset	Table A.3	**---**	**---**	**HR**	**HR**
4a	Certified tools and certified translators	Table A.3	**R**	**HR**	**HR**	**HR**
4b	Tools and translators: increased confidence from use	Table A.3	**HR**	**HR**	**HR**	**HR**

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

Figure 24: Extracts from IEC 61508-3 Table C.3 as referenced by IEC 60730. Highlighted techniques and measures are supported by the LDRA tool suite.

Programming languages

In selecting a "*suitable programming language*", IEC 61508-7[34] suggests that "*The programming language chosen should lead to an easily verifiable code with a minimum of effort and facilitate program development, verification and maintenance*".

Features which make verification difficult and therefore should be avoided are:

- unconditional jumps excluding subroutine calls,
- recursion,
- pointers, heaps or any type of dynamic variables or objects,

[34] IEC 61508-7:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 7: Overview of techniques and measures https://webstore.iec.ch/publication/5521

- interrupt handling at source code level,
- multiple entries or exits of loops, blocks or subprograms,
- implicit variable initialization or declaration,
- variant records and equivalence, and
- procedural parameters.

Tool qualification

IEC 60730 standard specifies the mechanism to provide evidence that the software tool chain can be relied upon, by once again referring to IEC 61508. The use of unproven tools implies detailed and thorough testing, which is a time consuming and costly process.

ZERTIFIKAT ◆ CERTIFICATE ◆ 認証証書 ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

TÜV SÜD Product Service

CERTIFICATE

No. Z10 17 12 84753 005

Holder of Certificate: LDRA Ltd.
Portside, Monks Ferry
Wirral
Merseyside CH41 5LH
UNITED KINGDOM

Factory(ies): 84753

Certification Mark:

Product: Software Tool for Safety Related Development

Model(s): LDRA tool suite
LDRArules
LDRAcover
LDRAunit
LDRAlite

Parameters: The certified tools, classified T2, fulfil the requirements for support tools according to IEC 61508-3 and EN 50128. The tools are qualified to be used in safety-related software development according to IEC 61508, EN 50128 and ISO 26262. It is suitably validated for use in safety-related development according to IEC 62304.

The test report is a mandatory part of this certificate.

Tested according to: IEC 61508-3:2010
ISO 26262-8:2011
EN 50128:2011
IEC 62304:2015

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

Test report no.: LW85043C

Valid until: 2022-12-20

Date, 2017-12-21 (Peter Weiss)

Page 1 of 1

737013

A1 / 04.11

TÜV SÜD Product Service GmbH · Zertifizierstelle · Ridlerstraße 65 · 80339 München · Germany

TÜV®

Figure 25: One of two TUV certificates applicable to the LDRA tool suite

In most cases, the most cost effective approach is therefore to use a tool that is already approved for the applied standard by an appropriate TÜV certifying organization (Figure 25). The required level of confidence in a software tool depends upon the circumstances of its deployment, with 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 item or element being developed, and the confidence in preventing or detecting such errors in its corresponding output.

A Tool Qualification Support Package (TQSP) can help to establish confidence in a TUV certified tool in the context of a particular development environment, in accordance with the specified class level.

Software modifications

These sections specify the steps to be followed during the modification of software. They provide guidance on the implementation of corrections, enhancements and adaptations of validated software, ensuring that the adherence to IEC 60730 for the resulting modified system is not compromised.

A software version management system is required at the module level, and all versions uniquely identified for traceability. Software modifications are required to be based on a modification request which details the proposed change and the reasons for it, and the hazards which may be affected.

IEC 61508-3 Table C.8[35] defines appropriate considerations. These include:

- the completeness and correctness of the modification with respect to its requirements,
- the freedom from introduction of intrinsic design faults,
- the avoidance of unwanted behavior,
- the verifiability and testability of the design, and
- the need for regression testing and verification coverage.

In this context, impact analysis is designed to determine whether a change or an enhancement to a software system has affected its overall functionality or has the potential to do so. Such an analysis will conclude that reverification will be required for only the changed software module in isolation, for all affected software modules, or for the complete system.

The level of re-verification required will be influenced by the number of software modules affected, the criticality of the affected software modules, and the nature of the change.

The facilities offered by the TBmanager component of the LDRA tool suite to illustrate the impact of changed requirements and the tool suite's capability to integrate with configuration and change control tools including Github[36], Apache® Subversion®[37], and Serena PVCS[38].

Conclusions

With its many sections, clauses and sub-clauses, IEC 60730 may at first seem intimidating, and its system of cross-referencing tables IEC 61508 and its annexes can make it difficult to follow. However, once broken down into digestible pieces, its principles offer sound guidance in the establishment of a high-quality software development process - not only leading up to initial product release but into maintenance and beyond. Such a process is paramount for the assurance of true reliability, quality, safety and effectiveness of programmable electronic components.

When supported by a complementary and comprehensive suite of tools for analysis and testing, the adoption of that process can smooth the way for development teams to work together to effectively develop and maintain large projects with confidence in their quality, simplifying the development process for Class B and Class C software in accordance with IEC 60730 (Figure 26).

[35] IEC 61508-3 Table C.8, " Properties for systematic safety integrity – Software modification"
[36] GitHub – Built for developers
https://github.com/
[37] Apache Subversion
https://subversion.apache.org/
[38] QBS Serena PVCS Version Manager
https://www.qbssoftware.com/serena-pvcs-version-manager_pvcsvm

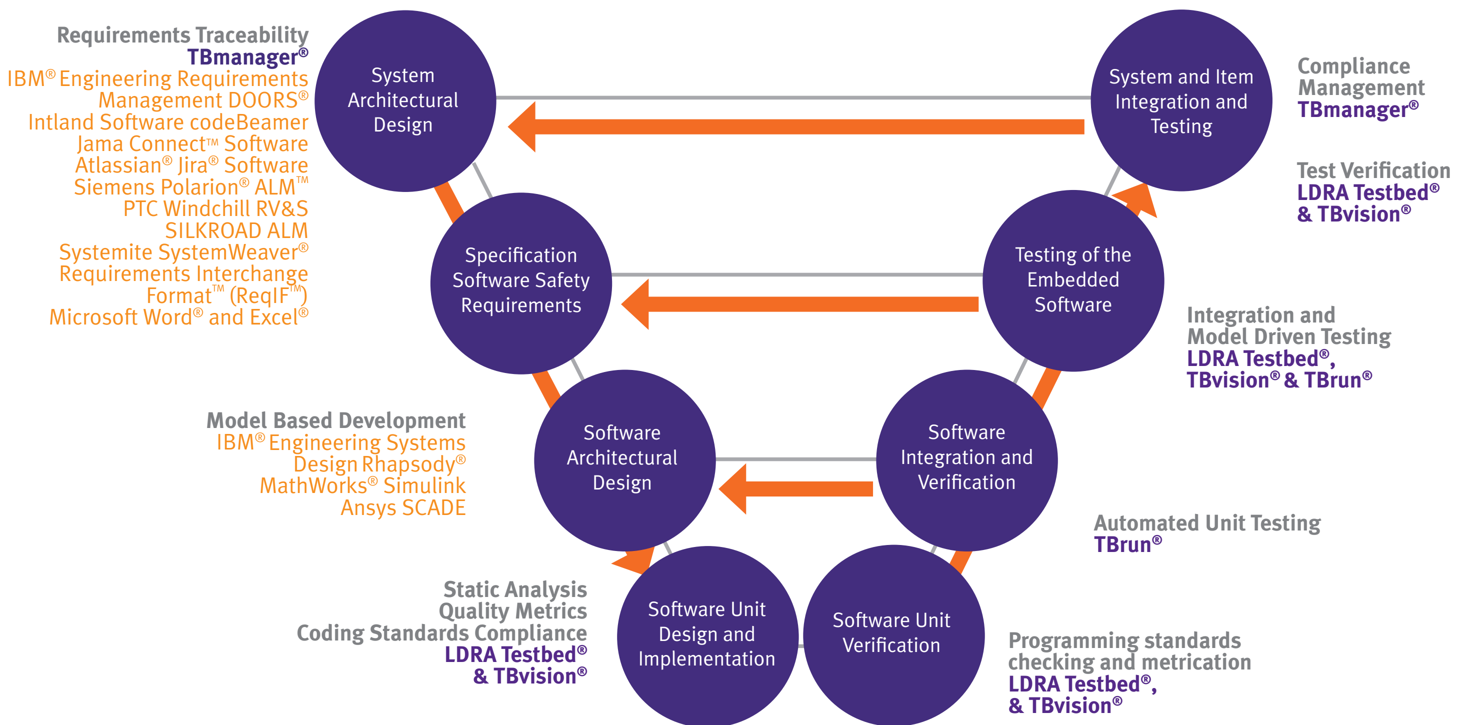

Figure 26: The role of automated tools in IEC 60730 compliant application development

References

MILITARY STANDARD: SOFTWARE DEVELOPMENT AND DOCUMENTATION (05 DEC 1994)
http://everyspec.com/MIL-STD/MIL-STD-0300-0499/MIL-STD-498_25500/

ISO/IEC 12207:1995. Information technology — Software life cycle processes. July 1995.
https://www.iso.org/standard/21208.html

ISO/IEC 15288:2008 Systems and software engineering — System life cycle processes
https://www.iso.org/standard/43564.html

IEC 1508: Functional Safety: Safety-Related Systems. August 1995.
https://ieeexplore.ieee.org/document/525946

IEC 61508-1:1998 Functional safety of electrical/electronic/programmable electronic safety-related systems
https://webstore.iec.ch/publication/19800

IEC 61508-1:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems
https://www.iecee.org/dyn/www/f?p=106:49:0::::FSP_STD_ID:5515

ISO 26262-1:2011 Road vehicles — Functional safety
https://www.iso.org/standard/43464.html

IEC 62304:2006+AMD1:2015 CSV Consolidated version Medical device software - Software life cycle processes
https://webstore.iec.ch/publication/22794

Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions
https://webstore.iec.ch/publication/3795

IEC 60730-1:2013 Automatic electrical controls
https://webstore.iec.ch/publication/3117

IEC: Functional Safety
https://basecamp.iec.ch/download/functional-safety-essential-to-overall-safety/

IEC 60335-1:2010 Household and similar electrical appliances - Safety - Part 1: General requirements
https://webstore.iec.ch/publication/1499

Controller for a washing machine (IEC 60730 / IEC 60335, 'Class B')
https://www.safetty.net/tt-design-examples/iec-60730-washer

IEC 60335-1:2010 Household and similar electrical appliances – Safety
https://webstore.iec.ch/publication/1499

IEC 60730-1:2013+AMD1:2015+AMD2:2020 CSV Consolidated version Automatic electrical controls
https://webstore.iec.ch/publication/66894

Exida resources – Systemic failure
https://www.exida.com/Resources/Term/systematic_failure

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

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

Ansys Scade
https://www.ansys.com/products/embedded-software

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

Siemens - Software Lifecycle Under Control
https://polarion.plm.automation.siemens.com/

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

IEC 61508-7:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 7: Overview of techniques and measures
https://webstore.iec.ch/publication/5521

GitHub – Built for developers
https://github.com/

Apache Subversion
https://subversion.apache.org/

QBS Serena PVCS Version Manager
https://www.qbssoftware.com/serena-pvcs-version-manager_pvcsvm

Industrial Safety starts with IEC/UL 60730 Standards.pdf by NXP
https://www.nxp.com/files-static/training_pdf/vFTF09_AZ125.pdf

Cypress: AN89056 - PSoC® 4 - IEC 60730 Class B and IEC 61508 SIL Safety Software Library
https://www.cypress.com/documentation/application-notes/an89056-psoc-4-iec-60730-class-b-and-iec-61508-sil-safety-software

Functional safety with 32-but microcontrollers
https://www.microchip.com/design-centers/32-bit/functional-safety

LDRA Technology Inc.
2540 King Arthur Blvd, Suite 228
Lewisville, Texas 75056
United States
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit No B-3, 3rd floor Tower B,
Golden Enclave. HAL Airport Road
Bengaluru
560017
India
Tel: +91 80 4080 8707
e-mail: india@ldra.com

www.ldra.com

LDRA

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

FM 26376