LDRA
Software Technology

Technical White Paper

Verification of Safety-Related Control System Software in Compliance with ISO 13849:2015

Working with Industry to Meet the Challenges of Achieving Cost-Effective Certification

www.ldra.com

Contents

Introduction

According to ISO 13849-1:2015 (Safety of machinery, Safety related parts of control systems, General principles for design)[1], a machine control system is defined as *"[a] system which responds to input signals from parts of machine elements, operators, external control equipment or any combination of these and generates output signals causing the machine to behave in the intended manner."* Given that the standard applies to *"all kinds of machinery"*, the machine under control could be anything from a toaster in a commercial kitchen, to a state-of-the-art industrial assembly robot.

Considering the sheer breadth of this remit, it is no surprise that the control systems applied to such machines may well deploy an amalgam of varying technologies, each reflected in their associated components – so that, for example, electronic PLCs (Programmable Logic Controllers), proportional pneumatic valves, and mechanical actuators could all have a part to play in a single machine control system. It also follows that the risk involved in operating machinery covered by the standard will range from negligible to highly hazardous.

ISO 13849-1:2015 is concerned only with safety-related parts of machine control systems (SRP/CS), and provides specific requirements for SRP/CS using programmable electronic systems. In order to ensure that the level of confidence is proportionate to the potential hazards associated with failure, the standard categorizes SRP/CS according to the demands placed upon them ("*Performance Level*" or "*PL*").

Related Standards

ISO 13849-1 is a European standard that is harmonized to the Machinery Directive[2], and replaced EN 954-1 (Safety of machinery, Safety related parts of control systems, General principles for design). EN 954-1[3] was completely withdrawn in 2011.

Because ISO 13849 embraces machinery and its electrical, hydraulic, pneumatic, and mechanical components, it references standards relating to such as pneumatic fluid power, mechanical interlocking devices, and circuit breaker power distribution. None of these are relevant to this white paper which focuses specifically on the objectives of ISO 13849-1:2015 as it relates to embedded software in the context of E/E/PE systems.

In this case, the position is confused somewhat by the existence of a second standard, EN 62061[4] (Safety of machinery, functional safety of safety-related electrical, electronic and programmable electronic control systems) which is also harmonized to the Machinery Directive. Unlike ISO 13849, IEC 62061 is based on the more generic standard IEC 61508 (Functional safety of electrical/electronic/programmable electronic safety-related systems).

To address this situation, a joint ISO-IEC technical committee (ISO/TC 199 & IEC/TC 44) was formed with the intent to merge ISO 13849 and IEC 62061 into a new standard, IEC/ISO 17305. However, there have proven to be difficulties with this process and an official release is yet to emerge. In the meantime, ISO 13849-1 states that "*When a safety-related control function is designed using one or more SRP/CS, each SRP/CS shall be designed either according to this part of ISO 13849 or according to IEC 62061 /IEC 61508)*." In other words, SRP/CS designed to an appropriate level in any of the three standards can be combined.

There is further complication in that for the most demanding applications, ISO 13849-1:2015 defers its software development lifecycle to that of IEC 61508 which unfortunately uses quite different terminology. Although this white paper does reference IEC 61508, it is recommended that the LDRA white paper specific to that standard is also studied.

The principles of risk assessment and risk reduction promoted by ISO 13849 are defined in the standard ISO 12100:2010 (Safety of machinery -- General principles for design -- Risk assessment and risk reduction[5]).

[1] ISO 13849-1:2015 Safety of machinery — Safety-related parts of control systems Paragraph 3.1.32

[2] DIRECTIVE 2006/42/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 17 May 2006 on machinery, and amending Directive 95/16/EC https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2006:157:0024:0086:EN:PDF

[3] BS EN 954-1:1997 Safety of machinery. Safety related parts of control systems. General principles for design

[4] BS EN 62061:2005+A2:2015 Safety of machinery. Functional safety of safety-related electrical, electronic and programmable electronic control systems

[5] ISO 12100:2010 Safety of machinery -- General principles for design -- Risk assessment and risk reduction

Performance Levels

The concept of performance levels in ISO 13849 is somewhat analogous to the Safety Integrity Levels (SILs) described by IEC 61508, even though the two standards evolved independently of one another (Figure 1). According to ISO 13849 there is no equivalent to SIL 4 "*since SIL 4 is dedicated to catastrophic events possible in the process industry*".

PL (ISO 13849)	SIL (IEC 61508-1, for information) high/continuous mode of operation
a	No correspondence
b	1
c	1
d	2
e	3
(ISO13849 cannot be used	4

Figure 1: Correlation between ISO 13849 Performance Levels, and IEC 61508 Safety Integrity Levels[6]

ISO 13849 requires that "*The SRP/CS shall be designed and constructed so that the principles of ISO 12100*[7] *are fully taken into account*". ISO 12100 specifies basic terminology, principles and a methodology for achieving safety in the design of machinery, and principles of risk assessment and risk reduction to help designers in achieving this objective. These principles are based on statistical techniques used to quantify knowledge and experience of the design, use, incidents, accidents and risks associated with machinery.

For example, a design team may opt to use the Hazard Rating System; an approach involving the quantification of the likely severity of harm, the number of people affected, the likelihood of an accident, and how frequently the risk occurs. This latter parameter will, in turn, be based on the reliability of the components, devices and circuitry.

The result of this calculation represents the probability of a dangerous failure over time, which is then categorised as a performance level (PL). The standard defines five performance levels, ranging from the least hazardous PL a to the most hazardous PL e.

The Performance Level is critical to the development of software development for the system, because the higher the PL rating, the greater the required overhead in terms of verification and validation. Conversely, increasing the verification and validation activity can minimize that overhead by achieving a lower target "*required performance level (PL)*".

The Software Development Lifecycle

ISO 13849 requires that "*All lifecycle activities of safety-related embedded or application software shall primarily consider the avoidance of faults introduced during the software lifecycle… The main objective of the following requirements is to have readable, understandable, testable and maintainable software.*"[8] Figure 2 is a reproduction of the simplified V-model referenced by ISO 13849, superimposed with an illustration of how the LDRA tool suite and other complementary tools can be applied within the process.

[6] Duplicated from ISO 13849-1:2015 Table 3, Relationship between performance level (PL) and safety integrity level (SIL)

[7] ISO 12100:2010 Safety of machinery -- General principles for design -- Risk assessment and risk reduction

[8] Extract from ISO 13849-1:2015 section 4.6.1

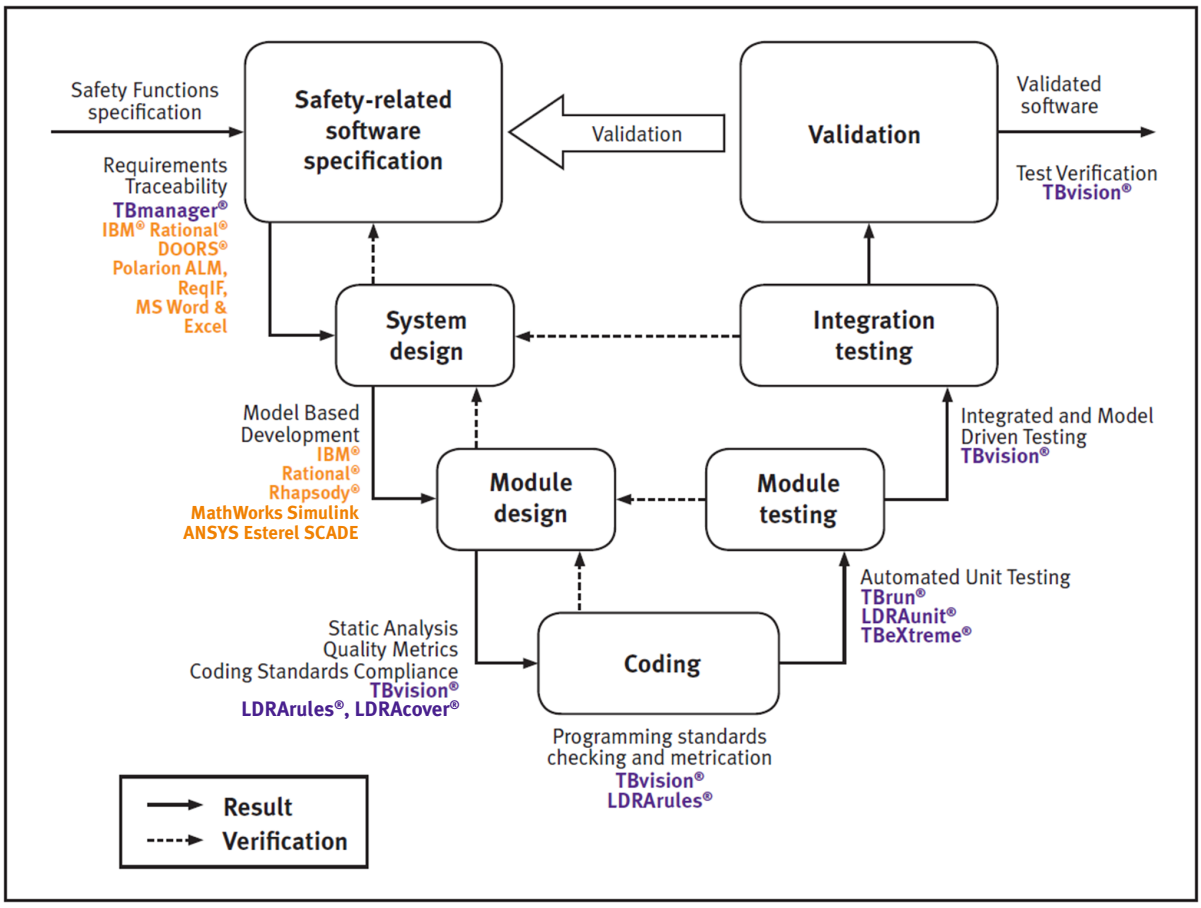

Figure 2: Mapping the capabilities of the LDRA tool suite and complementary tools to the ISO 13849-1:2015 Simplified V-model[9]

The simplified V-model applies to all Performance Levels, and so is used as a reference in this white paper.

For the most safety critical applications, ISO 13849 defers to the E/E/PES safety lifecycle requirements laid down in section 7 of IEC 61508- Part 3: Software requirements[10]. In addition to having no equivalent to SIL 4 from that standard, ISO 13849 also states "*For safety-related embedded software for components with PL_r = e, see IEC 61508–3:1998, Clause 7.*"

In other words, much of the development lifecycle specified in ISO 13849 applies only to PL d or less – or, in IEC 61508 terms, SIL 2 or less. This cross-referencing of standards is not helped by a mismatch in terminology, particularly in that IEC 61508 calls its instructions to the development team "*objectives*" whereas ISO 13849 refers to them as "*requirements*". Both IEC 61508 and IEC 13849 also use the word "*requirements*" for the project requirements – that is, the requirements that the project designers must fulfil for the product to achieve acceptable function safely.

For clarity, this white paper avoids the overloaded term, and applies the IEC 61508 approach to distinguish between the objectives of the standards, and the requirements of the project.

Less demanding applications follow the development life cycle specified in ISO 13849 itself, which subdivides them into "*Safety-related embedded software (SRESW)*" and "*Safety-related application software (SRASW)*". Each has slightly different sets of objectives.

The main body of the ISO 13849 standard lists the processes associated with embedded and application safety-related software, and cross-references them to the V-model. An example project in Annex J of the standard helps to place many of these activities into context, and Figure 3 shows a reproduction of table J.1 from that annex for reference.

[9] Based on ISO 13849-1:2015 Figure 6 — Simplified V-model of software safety lifecycle

[10] IEC 61508-3:2010 (Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 3: Software requirements)

Development activity	Verification activity	Associated documentation
Machine aspect: Identification of the functions involving the SRP/CS	Identification of safety-related functions	"Safety-related specification for machine-control"
Architecture aspect: Definition of the control architecture with sensors and actuators	Comments upon safety characteristics of chosen components	"Definition of the control architecture"
Software specification aspect: Transcription of machine functions into software functions	Re-reading of the descriptions (see J.3)	"Software descriptions"
Software architecture aspect: to detail the functions into functional blocks	Definition of critical blocks which are subject of greater review and validation effort	"Function block modelling"
Encoding aspect: Encoding according to the programming rules (see J.4)	RE-reading of the code Verifiction of functions and compliance with rules	"Encoding comments in the code" "Encoding re-reading sheets"
Validation aspect: Making of test scenarios: operation aspect of functions behaviour-on-failure aspect	Verification of the test covering Verification of the test results	"Correspondence matrix" which cross-references specification paragraphs and tests "Test sheets" comprising test scenario and comments upon results achieved

Figure 3: ISO 13849-1:2015 Table J1 - Activities and documents within software safety lifecycle

Safety-Related Software Specification

The first step in the simplified V-model concerns the definition of a software related specification. The V-model illustrates the need for each step in the process to be traceable to the next, as implied by the verification arrows during the lifecycle, and the validation step at its end.

For more critical projects adhering to the IEC 61508:2010 lifecycle, bi-directional traceability is specified by that standard as an explicit objective in its Annex A tables.

Achieving a format that lends itself to bi-directional traceability will help to achieve compliance with the standard. Bigger projects, perhaps with contributors in geographically diverse locations, are likely to benefit from an application lifecycle management tool such as IBM® Rational® DOORS®[11], Siemens® Polarion® PLM®[12], or more generally, similar tools offering support for standard Requirements Interchange Formats[13]. Smaller projects can cope admirably with carefully worded Microsoft® Word® or Microsoft® Excel® documents, written to facilitate links up and down the development process model.

Bi-Directional Traceability

It would be easy to dismiss the task of tracing between the development lifecycle phases as trivial, but their combined effect on project management overhead can be significant.

For instance, consider an unexpected change of requirement imposed by a customer. What is impacted? Which requirements? What elements of the code design? What code needs to be revised? And which parts of the software will require re-testing?

[11] http://www-03.ibm.com/software/products/en/ratidoor

[12] https://polarion.plm.automation.siemens.com/

[13] http://www.omg.org/spec/ReqIF/

The most effective way to ensure that the project is not thrown off course by such eventualities is to maintain Bidirectional Traceability of Requirements[14] (Figure 4). In this diagram, when requirements are managed well, traceability can be established forwards from the outline requirements, through the detailed requirements and on to the work products – and they can similarly be traced backwards. Such bidirectional traceability helps determine that all source requirements have been completely addressed and that all lower level requirements can be traced to a valid source, and that there is no spurious source code that is surplus to requirements. Requirements traceability can also cover the relationships to other entities such as intermediate and final work products, changes in design documentation, and test plans.

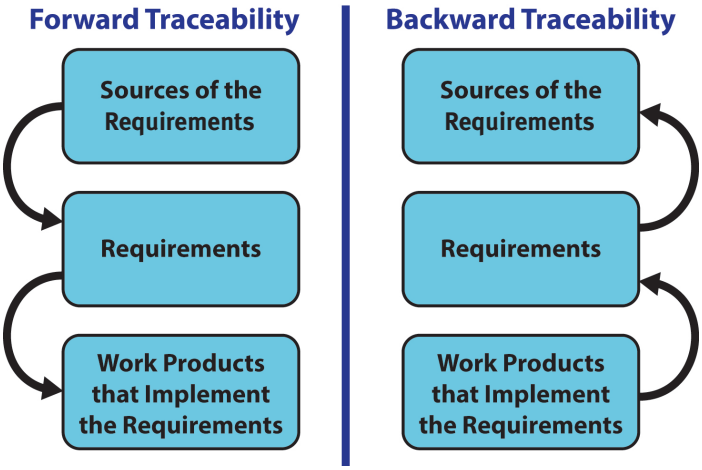

Figure 4: An Illustration of the principles of Bidirectional Traceability

Requirements rarely remain unchanged throughout the lifetime of a project, and that can turn the maintenance of a traceability matrix into an administrative nightmare. Furthermore, connected systems extend that headache into maintenance phase, requiring revision whenever a vulnerability is exposed.

A requirements traceability tool alleviates this concern by automatically maintaining the connections between the requirements, development, and testing artefacts and activities. Any changes in the associated documents or software code are automatically highlighted such that any re-testing can be dealt with accordingly (Figure 5).

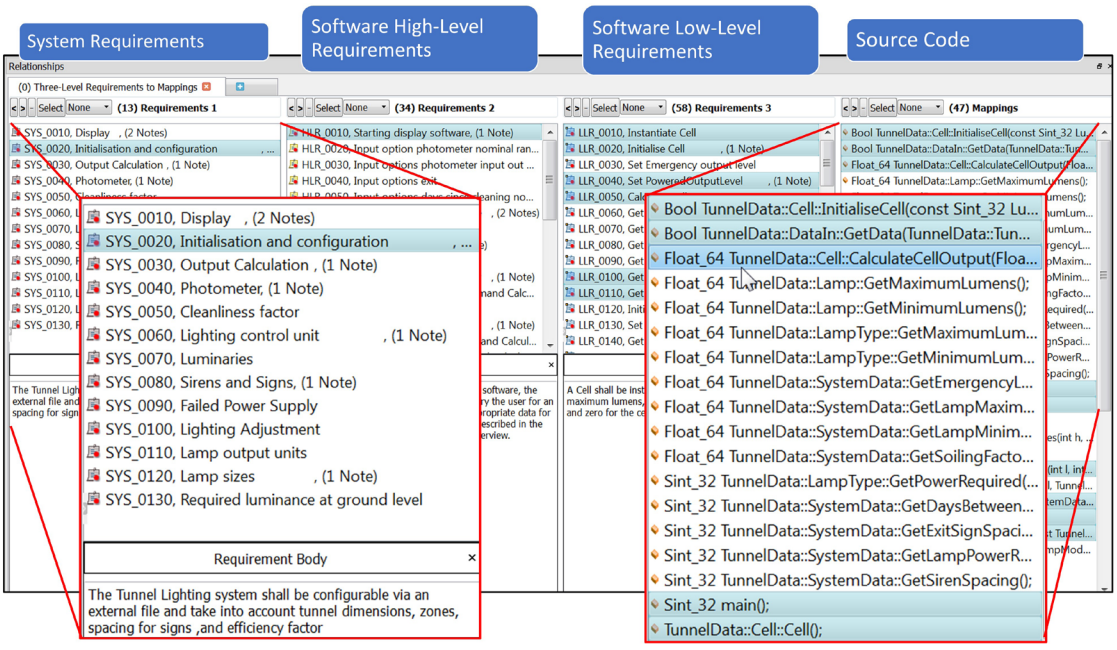

Figure 5: Automating requirements traceability with the TBmanager component of the LDRA tool suite

[14] http://www.westfallteam.com/Papers/Bidirectional_Requirements_Traceability.pdf Bidirectional Requirements Traceability, Linda Westfall

System Design

Irrespective of the required Performance Level, software design activity involves the definition of the structural components of the software, their externally visible properties, and the relationships between them. Any software component behaviour that can affect other components should be described, such that all software requirements can be implemented by the specified software items. This is generally verified by technical evaluation.

In the specific case of projects adhering to the IEC 61508 development lifecycle, the principles of software design remain the same although the level of verification work required is specified in more detail. Again, there is a terminology mismatch with IEC 61508 referencing "*software architecture*", a concept encompassed within the System Design phase of ISO 13849 which uses the term "*architecture*" in the context of a "*Function Blocks*" concept, which will be broadly familiar to anyone familiar with PLC ladder logic.

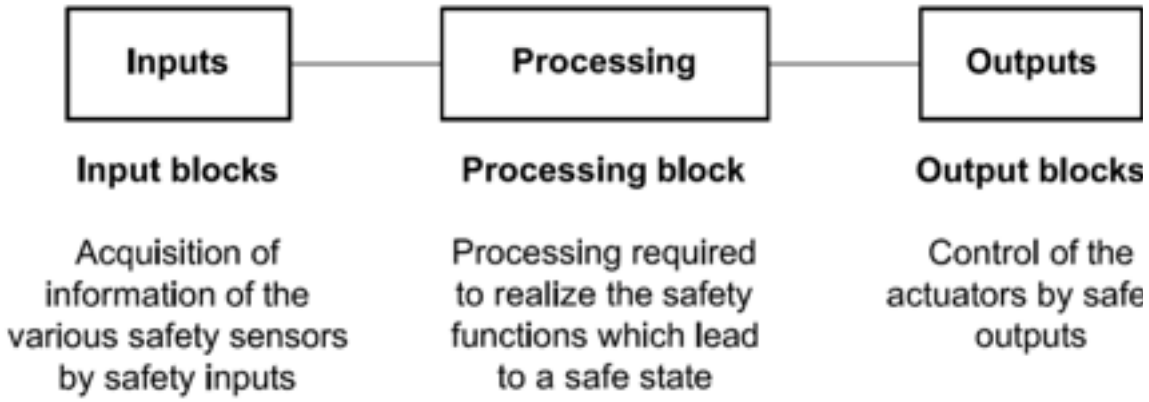

Figure 6: ISO 13849 general architecture model[15]

These function blocks are used to ensure that the system adheres to a "*Least Privilege*" principle[16], which in practical terms is highly beneficial because the resulting modularization and domain separation allows "*critical blocks*" to be identified. These are subjected to greater review and validation effort than the remainder[17].

Since the system design is based on the software specification, developing it means defining the interfaces between the software items that will implement the requirements. Many of these software elements will be created by the project, but some may be brought in from other projects, possibly in the form of third-party libraries. IEC 61508 highly recommends the use of "*trusted/verified software elements*[18]". Note that many components of the LDRA tool suite are as applicable to legacy code, as they are to code that is newly written.

If a model-based approach is taken to system design - for example, using MathWorks® Simulink®[19], IBM® Rational® Rhapsody®[20], or ANSYS® SCADE[21] then a tool suite that is integrated with the chosen modelling tools will make the analysis of generate code and traceability to the models far more seamless.

[15] Based on ISO 13849-1:2015 Figure 7 — General architecture model of software

[16] https://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP

[17] ISO 13849-1:2015 Table J.1 — Activities and documents within software safety lifecycle - Verification activity

[18] IEC 61508-3 Table A.2 – Software design and development – software architecture design – Technique/measure 8

[19] https://uk.mathworks.com/products/simulink.html

[20] http://www-03.ibm.com/software/products/en/ratirhapfami

[21] http://www.ansys.com/products/embedded-software/ansys-scade-suite

Module Design

Once the specification and system design are defined and verified, module design can be built on this foundation. Module design specifies algorithms, data representations, and interfaces between different software units and data structures. Because implementation depends on module design, it is necessary to verify the module design before the activity is complete, generally by means of a technical evaluation of the module design as a whole, and verification of each software unit and its interfaces.

Where the IEC 61508 development lifecycle is applicable, the analogous phase to module design is known as “Detailed design and development”. IEC 61508 describes principles of software design that are broadly similar to those of IEC 13849, although the level of verification work required is specified in more detail in the former. In either case, a requirements traceability tool can help by verifying that all aspects of system design are traceable to module design, and vice versa.

Later in the development cycle, a tool suite can help by generating graphical artefacts suited to the review of the implemented design by means of walkthroughs or inspections. One approach is to prototype the software architecture in an appropriate programming language, which can also help to find any anomalies in the design. Graphical artefacts like call graph and flow graphs are well suited for use in the review of the implemented design by visual inspection (Figure 7).

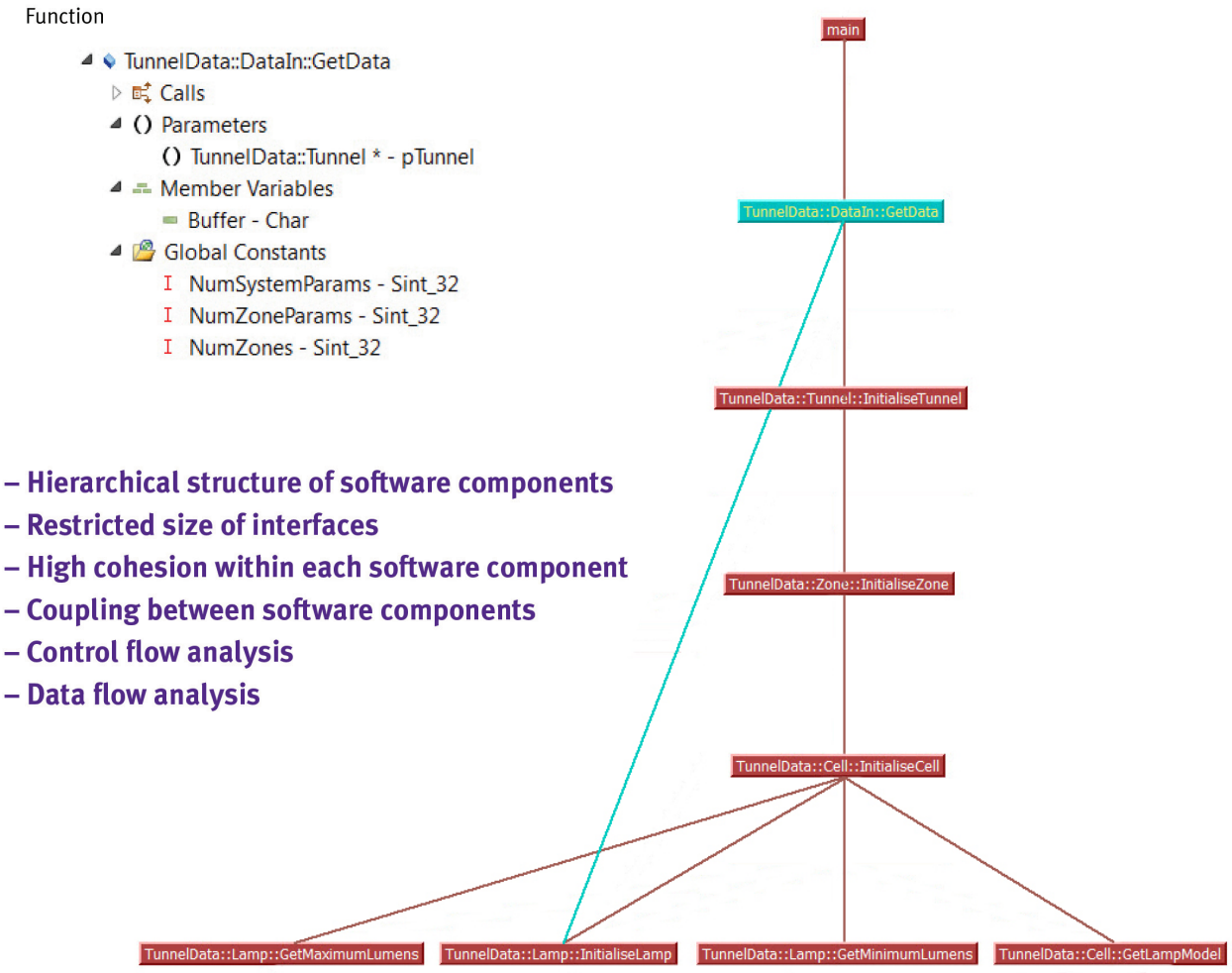

Figure 7: Diagrammatic representations of control and data flow generated from source code by the LDRA tool suite aid verification of implemented code versus software specification and design

Coding

Static Analysis

Static analysis, also called static code analysis, is a method of assessing the quality of software code by examining the code without executing it.

Both ISO 13849 (PLr c or d) and IEC 61508 (where applicable) require the use of coding standards, which are rule sets restricting the use of the chosen language to a preferred subset. In general, coding standards are used to specify a preferred coding style, aid understandability, apply language usage rules or restrictions, and manage complexity. The use of a static analysis tool is recommended to ensure that code complies with the nominated standard in a timely and cost-effective manner.

Verification tools such as the TBvision component of the LDRA tool suite largely offer support for a range of coding standards such as MISRA C and C++, JSF++ AV, HIS, CERT C, and CWE. The more sophisticated tools are able to operate in both "heavyweight" and "lightweight" modes, with the former able to confirm adherence to a very high percentage of the rules dictated by each standard, and the latter being quicker but unable to detect the more subtle violations. Such solutions will also support the creation of, and adherence to, in-house standards from both user-defined and industry standard rule sets.

Beyond the use of coding standards, automated static analysis can also make other valuable contributions, including the generation of code quality complexity metrics to help limit the complexity of source code and hence make it more reliable and maintainable, the enforcement of structured control flow, and the creation of data and control flow diagrams. Specific examples of how automated static analysis can aid adherence to ISO 13849 objectives are detailed in Figure 8.

Description	ISO 13849-1:2015 reference
Use of coding standards	*"4.6.2 For SRESW for components with PL_r c or d, the following additional measures shall be applied ..... use of design and coding standards"*
Limiting module size	*"4.6.2 For SRESW for components with PL_r c or d, the following additional measures shall be applied... "limited module sizes with fully defined interfaces"* *"4.6.3 For SRASW for components with PL_r from c to e, the following additional measures... are required or recommended....function blocks of limited size of coding".*
Control flow analysis	*"4.6.3 For SRASW for components with PL_r from c to e, the following additional measures... are required or recommended ... verification should be by control and data flow analysis for PL = d or e."*
Data flow analysis	*"4.6.3 For SRASW for components with PL_r from c to e, the following additional measures... are required or recommended ... verification should be by control and data flow analysis for PL = d or e."* *"J.4 a Examples of programming rules ... c) ...The function blocks shall not modify the global variables*
Structured programming	*"4.6.2 For SRESW for components with PL_r c or d, the following additional measures shall be applied... "modular and structured programming"*
Structure diagrams	*"J.4 Examples of programming rules ... a. The programming should be structured so as to display a consistent and understandable general skeleton allowing the different processings to be easily localized*
Defensive programming	*"4.6.3 For SRASW for components with PL_r from c to e, the following additional measures... are required or recommended ... use of techniques for detection of external failure and for defensive programming"*
Detection of conditions causing systematic error	*"4.6.3 For SRASW for components with PL_r from c to e, the following additional measures... are required or recommended... Technical features which detect ... such as data type mismatch, ambiguous dynamic memory allocation, incomplete called interfaces, recursion, pointer arithmetic ... shall be used*

Figure 8: Examples of ISO 13849 objectives that can be met through the application of LDRA static analysis tools

Again, for those projects where it applies, IEC 61508 is more detailed in its specification of what is needed. Figure 9 is an annotated duplicate of Table B.1 from that standard.

Technique/Measure		Ref	SIL			
			1	2	3	4
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 SIL. "R" The method is recommended for this SIL. "---" The method has no recommendation for or against its usage for this SIL.						

Figure 9: Copy of IEC 61508 Table B.1[22], with static analysis techniques provided by the LDRA static analysis tools highlighted

Module Testing

ISO 13849 is quite explicit in its objectives for module (or "*unit*") testing – that is, for tests which execute a subsection of the code. For example, part 2 ("*Validation*") states:

"In general, software can be considered a 'black box' or 'grey box'... and validated by the black- or grey-box test, respectively."

Depending on the PL_r ... the tests should include:
— black-box testing of functional behaviour and performance (e.g. timing performance),
— additional extended test cases based upon limit value analyses, recommended for PL d or e,
— I/O tests to ensure that the safety-related input and output signals are used properly, and
— test cases which simulate faults determined analytically beforehand, together with the expected response, in order to evaluate the adequacy of the software-based measures for control of failures."

More examples of how automated dynamic analysis can aid adherence to ISO 13849 objectives are detailed in Figure 10.

Description	ISO 13849 reference
Functional and black-box testing	ISO 13849-1 *"4.6.3 For SRASW for components with PL_r from c to e, the following additional measures... are required or recommended...the appropriate validation method is black-box testing of functional behaviour and performance criteria"* ISO 13849-1"G.4 *"In addition, black-box testing should be applied"* ISO 13849-2"9.5 *"black-box testing of functional behaviour and performance"*
Automated boundary-condition test generation	ISO 13849-1 *"4.6.3 For SRASW for components with PL_r from d to e, the following additional measures... are required or recommended... test case execution from boundary value analysis is recommended;*
Fault injection tests	*ISO 13849-2 "9.1 Analysis and testing ... When validating by testing, the tests should include, as appropriate ..... fault injection tests into a production sample"* *ISO 13849-2 "9.7 Validation of combination of safety-related parts ... for redundant systems, fault injection tests relating to combination/integration.*
Model-driven tests	*ISO 13849-2 "9.1 Analysis and testing... a simulation of control system behaviour in the event of a fault, e.g. by means of hardware and/or software models."*

Figure 10: Examples of ISO 13849 objectives that can be met through the application of LDRA dynamic analysis tools

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

Unit test tools such as the TBrun component of the LDRA tool suite (Figure 11) provide a graphical user interface for unit test specification which is used to create tests according to the defined specification and to present a list of all defined test cases with appropriate pass/fail status. The ability to automatically present the graphical presentation of control flow graphs, and to create test harnesses, stub functions, and cover for missing member or global variables means that unit test execution and interpretation becomes a much quicker and easier process, requiring a minimum of specialist knowledge. By extending the process to the automatic generation of test vectors, the tools can also provide a straightforward means to analyse boundary values without creating each test case manually. Test sequences and test cases are retained so that they can be repeated ("*regression tested*"), and the results compared with those generated when they were first created.

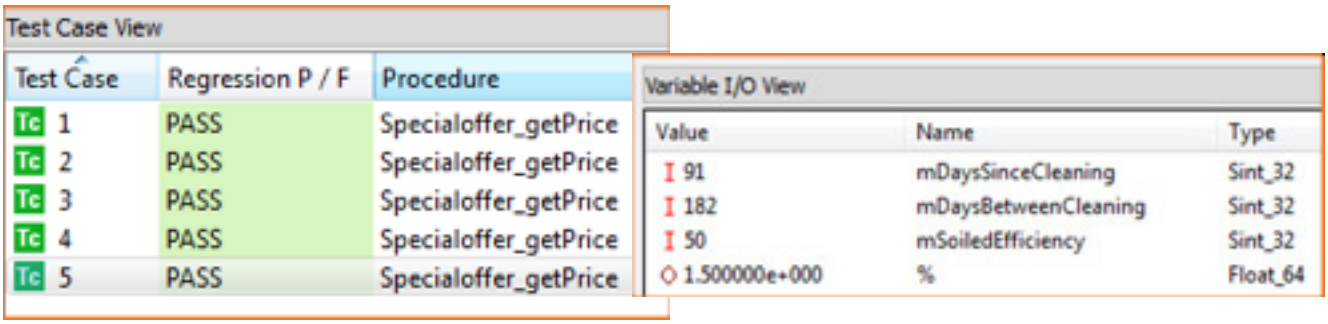

Figure 11: Test Case Automation with the TBrun component of the LDRA tool suite

"*Black box testing*" is a method of software testing that examines the functionality of an application without peering into its internal structures or workings, whereas in this context, "*Grey box testing*" demands some knowledge of the internal structure, which includes access to the documentation of internal data structures as well as the algorithms used.

Where the IEC 61508 development lifecycle is in use, more precise definitions of the type of unit tests to be implemented are provided, and structural test coverage is also demanded (Figure 12).

	Technique/Measure	Ref	SIL			
			1	2	3	4
1	Test case execution from boundary value analysis	C.5.4	R	HR	HR	HR
2	Test case execution from error guessing	C.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	C.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	Structural test coverage (entry points) 100 % **	C.5.8	HR	HR	HR	HR
7b	Structural test coverage (statements) 100 %**	C.5.8	R	HR	HR	HR
7c	Structural test coverage (branches) 100 %**	C.5.8	R	R	HR	HR
7d	Structural test coverage (conditions, MC/DC) 100 %**	C.5.8	R	R	R	HR

"HR" The method is highly recommended for this SIL.
"+" The method is recommended for this SIL.
"---" The method has no recommendation for or against its usage for this SIL.
** Where 100% coverage cannot be achieved (e.g. statement coverage of defensive code), an appropriate explanation should be given

Figure 12: Copy of IEC 61508 Table B.2[23], with dynamic analysis techniques provided by the LDRA tool suite highlighted

In addition to showing that the software functions correctly, dynamic analysis is also used to generate structural coverage metrics to provide evidence of test completeness and to demonstrate that there is no unintended functionality. The LDRA tool suite is capable of providing statement, branch and MC/DC coverage from either unit or system test, on the target hardware platform if required (Figure 13).

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

System and unit test can also operate in tandem, so that (for instance) coverage can be generated for most of the source code through a dynamic system test and complemented using unit tests to exercise normally inaccessible code, such as defensive constructs.

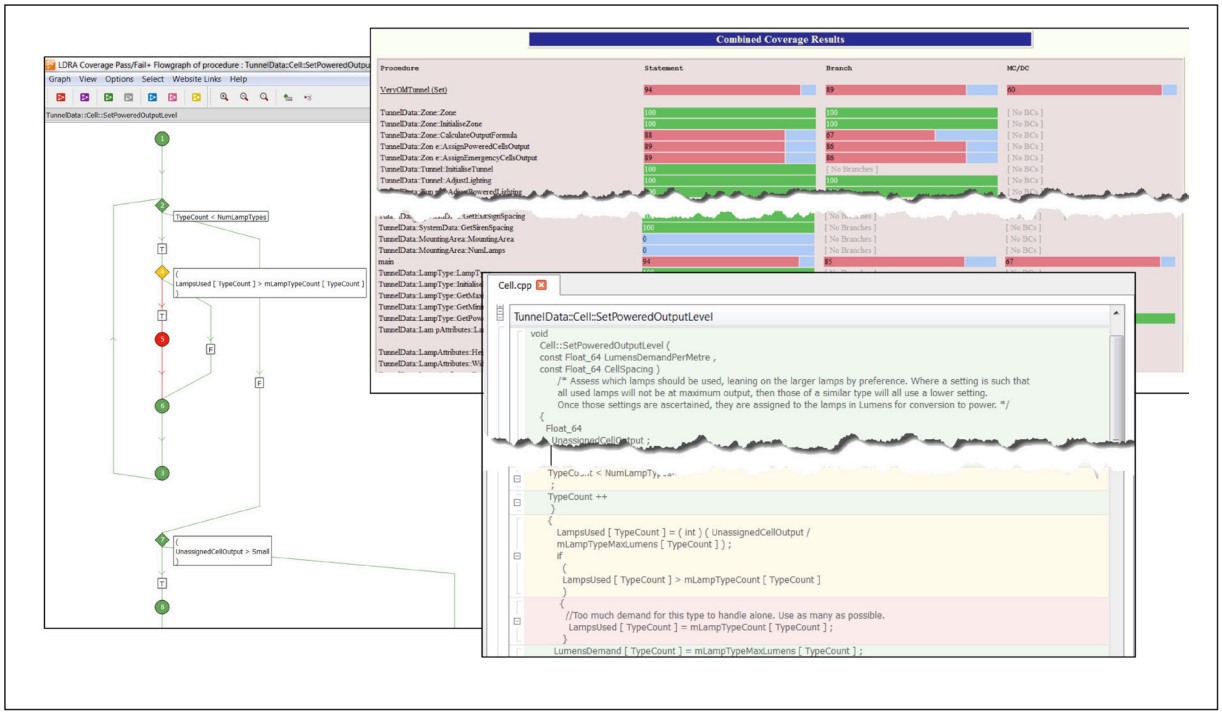

Figure 13: Examples of representations of structural coverage within the LDRA tool suite

Integration

Software integration testing focuses on the transfer of data and control across a software module's internal interfaces and external interfaces such as those associated with SRP/CS hardware, operating systems, and third party software applications and libraries. This activity requires the integration of software units and/or other elements into ever larger aggregated items, ultimately verifying that the resulting integrated system behaves as intended.

Integration of software functions into the SRP/CS application, and integration of multiple SRP/CS into a complete system, carry similar implications as far as the software is concerned (Figure 10).

Description	ISO 13849 reference
Validated function blocks	*ISO 13849-1 "4.6.3 …Whenever reasonable and practicable, validated function block (FB) libraries should be used…"*
Avoidance of systemic failures	*SO 13849-2 "G.4 Measures for avoidance of systematic failures during SRP/CS integration - In addition, black-box testing should be applied, taking into account the complexity of the SRP/CS and its PL."*
Validation of combination of safety-related parts	*ISO 13849-2 "9.7 Where the safety function is implemented by two or more safety-related parts, validation of the combination — by analysis and, if necessary, by testing — shall be undertaken to establish that the combination achieves the performance level specified in the design."* *ISO 13849-2 "9.7 … for redundant systems, fault injection tests relating to combination/integration."*

Figure 14: Examples of ISO 13849 objectives related to system integration

Once again, for projects required to adhere to the IEC 61508 development lifecycle, the objectives are more extensive and more precisely defined (Figure 15).

	Technique/Measure	Ref	SIL			
			1	2	3	4
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
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 SIL.
"R" The method is recommended for this SIL.
"---" The method has no recommendation for or against its usage for this SIL.

Figure 15: Copy of IEC 61508 Table A.5[24], with Software module testing and integration techniques provided by the LDRA tool suite highlighted

As far as tooling is concerned, unit test tools such as the TBrun component of the LDRA tool suite can use exactly the same mechanisms (and even the same tests) to validate the functionality of software functions in combination, as it can to test them in isolation. User interfaces allow the test engineer to specify at what point in the call tree they would like the system to be "*stubbed*", giving ultimate control on how the code is exercised.

Integration testing can also be used to demonstrate program behaviour at the boundaries of its input and output domains and confirms program responses to invalid, unexpected, and special inputs. The program's actions are revealed when given combinations of inputs or unexpected sequences of inputs are received, or when defined timing requirements are violated. The test requirements in the plan should include, as appropriate, the types of white box testing and black box testing to be performed as part of integration testing.

Testing tools need to have the capability to provide dynamic structural coverage analysis, both at system test level and at unit test level, which is a mechanism to show which parts of the code base have been exercised during testing. The coverage data derived from unit and system test can be combined to provide the most effective way of working for the particular needs of a development project. A common approach is to operate them in tandem, so that (for instance) coverage can be generated for most of the source code through a dynamic system test, and complemented by unit tests to exercise defensive code and other previously uncovered elements.

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

Testing of changed source code can again be performed using regression testing. It is advisable to validate test cases as a matter of course and perhaps automatically, to ensure that any changed code has not affected proven functionality elsewhere. The requirements for structural coverage metrics like statement, branch, condition, procedure/function call, and data flow coverage vary depending on device classification, and all are provided by both the unit test and system test facilities within tools such as the LDRA tool suite.

Validation

Validation at the system level requires the manufacturer to verify the software's functionality by verifying that the requirements for the software have been successfully implemented. Software system testing demonstrates that the specified functionality exists in the system as it will be deployed, and that performance of the program is as specified.

Software system testing is in many ways the ultimate integration test, and comments relating to integration testing still apply here. Functional (black box) testing with no code coverage can also be performed although it might be desirable to use white box methods to more efficiently accomplish certain tests, initiate stress conditions or faults, or increase code coverage of the qualification tests.

Software system testing tests the integrated software and can be performed either in a simulated environment, on actual target hardware, or on the implemented SRP/CS.

This brings the narrative back to the earlier discussion of requirements traceability tools, such as the TBmanager component of the LDRA tool suite. A requirements traceability tool automatically maintains the connections between the requirements, development, and testing artefacts and activities. In short such a traceability tool ensures that requirements are correctly implemented at all times throughout the development lifecycle. System validation remains necessary, but will consist of providing evidence that requirements are fulfilled rather than checking to see whether they are.

The Connected System – a New Significance for System Maintenance

With the advent of the connected device and the Internet of Things, system maintenance takes on a new significance. For any connected systems, requirements don't just change in an orderly manner during development. They change without warning - whenever some smart Alec finds a new vulnerability, develops a new hack, or compromises the system. And they keep on changing throughout the lifetime of the device.

For that reason, the ability of next-generation automated management and requirements traceability tools and techniques to create relationships between requirements, code, static and dynamic analysis results, and unit- and system-level tests is especially valuable for connected systems. Linking these elements already enables the entire software development cycle to become traceable, making it easy for teams to identify problems and implement solutions faster and more cost effectively. But they are perhaps even more important after product release, presenting a vital competitive advantage in the ability to respond quickly and effectively whenever security is compromised. The handling of such change is highlighted by both ISO 13849 (Figure 16).

Impact Analysis

Description	ISO 13849 reference
SRESW modifications	*ISO 13849-1 "4.6.2 …impact analysis and appropriate software safety lifecycle activities after modifications…."*
SRESW modifications	*ISO 13849-1 "4.6.3 j) After modifications of SRASW, impact analysis shall be performed to ensure specification. Appropriate lifecycle activities shall be performed after modifications. Access rights to modifications shall be controlled and modification history shall be documented."*

Figure 16: Examples of ISO 13849 objectives related to system integration

Once again, for projects required to adhere to the IEC 61508 development lifecycle, the objectives are more extensive and more precisely defined (Figure 17).

	Technique/Measure	Ref	SIL			
			1	2	3	4
1	Impact analysis	C.5.23	HR	HR	HR	HR
2	Reverify changed software module	C.5.23	HR	HR	HR	HR
3	Reverify affected software modules	C.5.23	R	HR	HR	HR
4a	Revalidate complete system	Table A.7	...	R	HR	HR
4b	Regression validation	C.5.25	R	HR	HR	HR
5	Software configuration management	C.5.24	HR	HR	HR	HR
6	Data recording and analysis	C.5.2	HR	HR	HR	HR
7	Forward traceability between the Software safety requirements specification and the software modification plan (including reverification and revalidation)	C.2.11	R	R	HR	HR
8	Backward traceability between the software modification plan (including reverification and revalidation) and the software safety requirements specification C.5.23	C.2.11	R	R	HR	HR

”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.
Satisfied by the LDRA tool suite.

Figure 17: Copy of IEC 61508 Table A.8, with techniques and measures provided by the LDRA tool suite highlighted

Many software modifications will require changes to the existing software functionality – perhaps with regards to additional utilities in the software. In such circumstances, it is important to ensure that any changes made or additions to the software do not adversely affect the existing code.

A requirements traceability tool such as LDRA TBmanager® helps alleviate this concern by automatically maintaining the connections between the requirements, development, and testing artefacts and activities. In the example shown in Figure 18, suppose that a change is proposed to the System Level requirement “*Installation and configuration*”. The traceability established at development time between requirements, code and tests mean that the tool can show which parts of the code are impacted by the proposed change, as highlighted in the example.

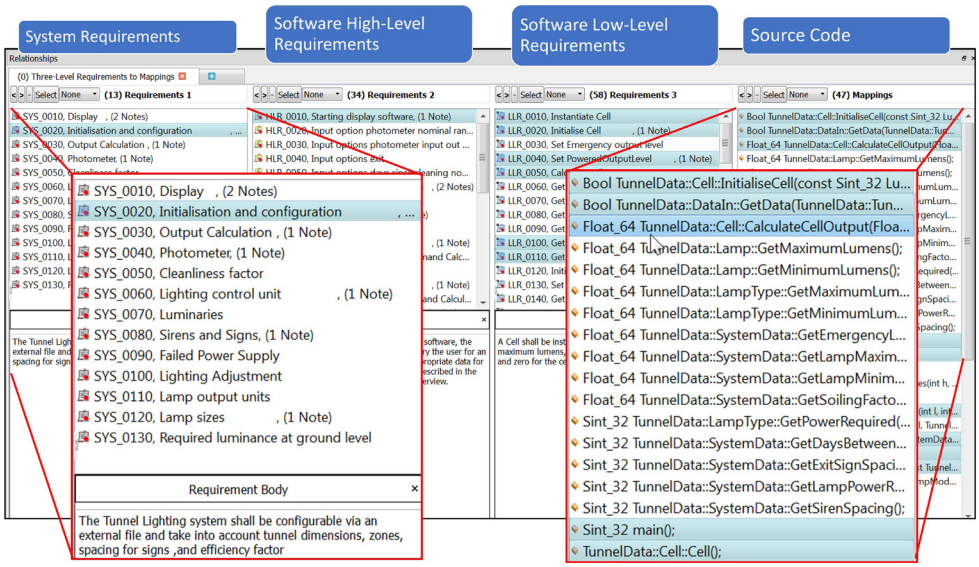

Figure 18: Identifying the impact of requirements change with the TBmanager component of the LDRA tool suite

The existing code as launched should also have undergone quality control measures in accordance with the ISO 13489 standard such as static analysis to assess whether coding standards have been met, and unit tests to confirm functionality of each code module.

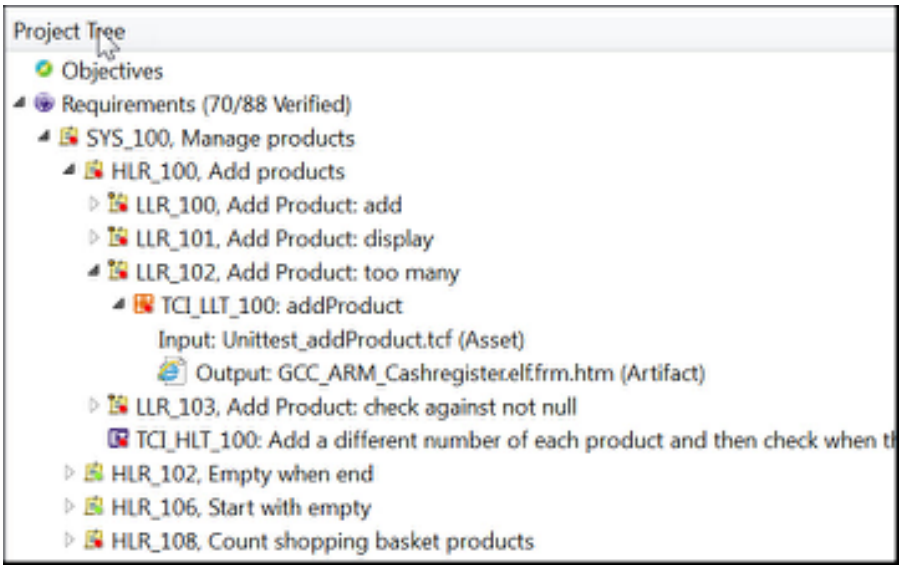

Figure 19: Showing functions requiring retest with the TBmanager component of the LDRA tool suite

Figure 19 shows a display from the TBmanager requirements traceability component of the LDRA tool suite. In this example, a system has been subject to a change request for the "*Add products*" requirement. Those parts of the system which are potentially affected by the change are easily identified by means of a red dot, whereas unaffected functions carry a green dot.

In the example, there is a test case file (.tcf) associated with each of four low-level requirements – "*add*", "*display*", "*too many*" and "*check against not null*". Those files retain the test vectors associated each of these low level requirements, meaning that they can all be re-run at the touch of a button and the code's functionality confirmed.

Of course, some tests will require change to reflect the new functionality dictated by the updated requirement, but in many cases it is enough to confirm that things work as they did previously. Automating the process in this way means that doing so requires minimal effort.

Development process artefacts such as coding standards compliance reports and code coverage reports are also retained and associated with requirements, so that as the tests are re-run ("*regression tested*") the artefacts are re-generated to go with them (Figure 20).

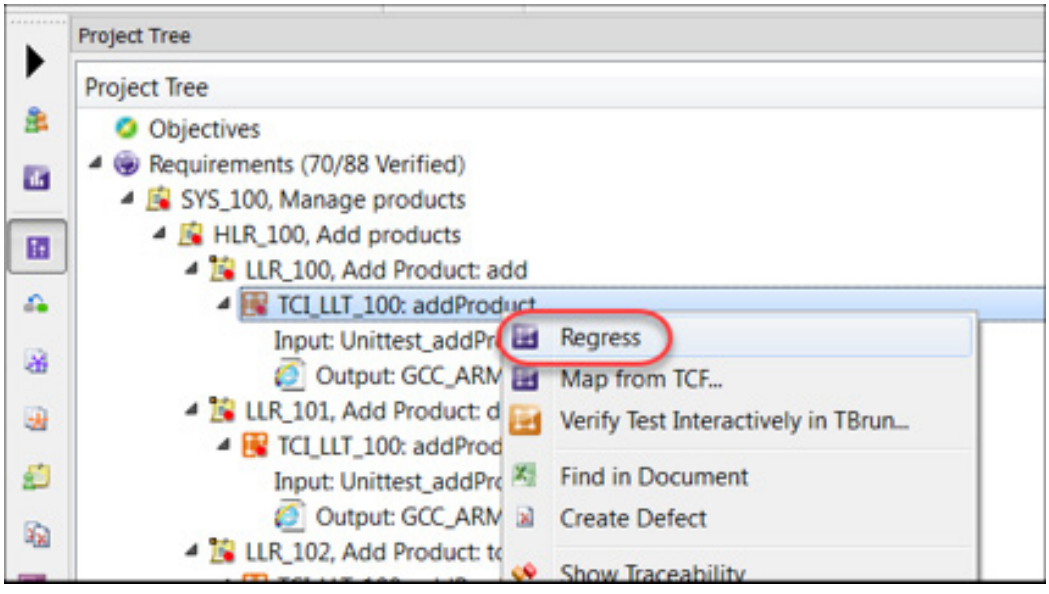

Figure 20: "Regression Testing" – Re-running unit tests to show that the functionality they describe still holds true, using TBmanager and TBrun components of the LDRA tool suite

Conclusion

ISO 13849 is a functional safety standard that can be used across a wide range of machinery which has evolved throughout the years, (via ISO/TR 13849-100:200, ISO/AWI 13849-1, ISO 13849-1:1999, ISO 13849-2:2003, ISO 13849-1:2006/Cor 1:2009, ISO 13849-2:2012, ISO 13849-1:2015) and which is applicable to all industries. Adherents to this standard are likely to include machine manufacturers in small, medium and large enterprises who may be building control devices for consumer goods, construction plant, market surveillance equipment or accident prevention machinery, or any other equipment with one or more safety considerations to contend with.

With its many sections, clauses and sub-clauses, IEC 13849 may at first seem intimidating, and its extensive referencing to other standards such as IEC 61508 for more critical applications 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 machinery. When supported by a complementary and comprehensive suite of tools for analysis and testing, it can smooth the way for development teams to work together to effectively develop and maintain large projects with confidence in their quality.

Works Cited

ANSYS SCADE Suite web site
http://www.ansys.com/products/embedded-software/ansys-scade-suite
Bidirectional Requirements Traceability, Linda Westfall
http://www.westfallteam.com/Papers/Bidirectional_Requirements_Traceability.pdf
EN 954-1 and ten reasons NOT to use it
http://www.machinebuilding.net/ta/to179.htm
Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 0: Functional safety and IEC 61508
http://www.iec.ch/functionalsafety/
IBM Rational DOORS web site
http://www-03.ibm.com/software/products/en/ratidoor
IBM Rational Rhapsody family web site
http://www-03.ibm.com/software/products/en/ratirhapfami
IEC/TR 62061-1 guide to application of ISO 13849-1 and IEC 62061
http://www.machinebuilding.net/n/n1916.htm
Introduction to ISO 13849-1 Safety Standard
https://www.robotics.org/blog-article.cfm/Introduction-to-ISO-13849-1-Safety-Standard/78
ISO 12100:2010 Safety of machinery -- General principles for design – Risk assessment and risk reduction
ISO 13849-1, Safety of machinery – Safety-related parts of control systems – Part 1 General principles for design and Part 2 Validation
MathWorks SIMULINK web site
https://uk.mathworks.com/products/simulink.html
Object Management Group Requirements Interchange Format web site
http://www.omg.org/spec/ReqIF/
Siemens Polarion ALM web site
https://polarion.plm.automation.siemens.com/
Techtarget definition “Principle of Least Privilege” (POLP)
https://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP
Working towards a merger of ISO 13849-1 and IEC 62061
http://www.machinebuilding.net/ta/to299.htm

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

LDRA Technology Inc.
2540 King Arthur Blvd, 3rd Floor, 12th Main Lewisville Texas 75056
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit B-3, Third floor Tower B, Golden Enclave
HAL Airport Road Bengaluru 560017
Tel: +91 80 4080 8707
e-mail: india@ldra.com

LDRA tool suite and ISO 13849 v2.0 09/18