# 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

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

## 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.*"

Considering the sheer breadth of this remit, it is no surprise that the control systems applied to such machines may well deploy a combination of various technologies, each reflected in their associated components. It also follows that the risk involved in operating machinery covered by ISO 13849-1 will range from negligible to highly hazardous. In order to ensure that the level of confidence is proportionate to the potential hazards associated with failure, the standard categorizes safety-related parts of machine control systems (SRP/CS) according to the demands placed upon them ("*Performance Leve*l" or "*PL*").

ISO 13849-1 is one of two standards that are harmonized to the Machinery Directive[2], with EN 62061[3] (Safety of machinery, functional safety of safety-related electrical, electronic and programmable electronic control systems) covering similar ground. A third standard IEC/ISO 17305 to merge the two is yet to be published. In the meantime, ISO 13849-1 suggests that SRP/CS designed to an appropriate level in any of the standards ISO 13849, IEC 62061 and IEC 61508 can be combined.

## Performance Levels

| PL ISO 13849 | SIL IEC 61508 |
|---|---|
| a | – |
| b | 1 |
| c | 1 |
| d | 2 |
| e | 3 |
| – | 4 |

The concept of performance level in ISO 13849 is somewhat analogous to the Safety Integrity Levels (SILs) described by IEC 61508 (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.*"

ISO 13849 requires that SRP/CS are designed and constructed according to statistically-based principles laid out in ISO 12100[5]. These provide the basis for the derivation of a performance level (PL). The standard defines five performance levels, ranging from the least hazardous PL to the most hazardous PL e.

Increasing the verification and validation activity can help to lower the target "*required performance level (PL$_r$).*"

*Figure 1: Correlation between ISO 13849 Performance Levels and, IEC 61508 Safety Integrity Levels*[4]

## The Software Development Lifecycle

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.

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. Much of the development lifecycle specified in ISO 13849 therefore applies only to PL d or less, and the standard further subdivides them into "*Safety-related embedded software (SRESW)*" and "*Safety-related application software (SRASW).*" Each has slightly different sets of objectives.

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

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

[5] 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

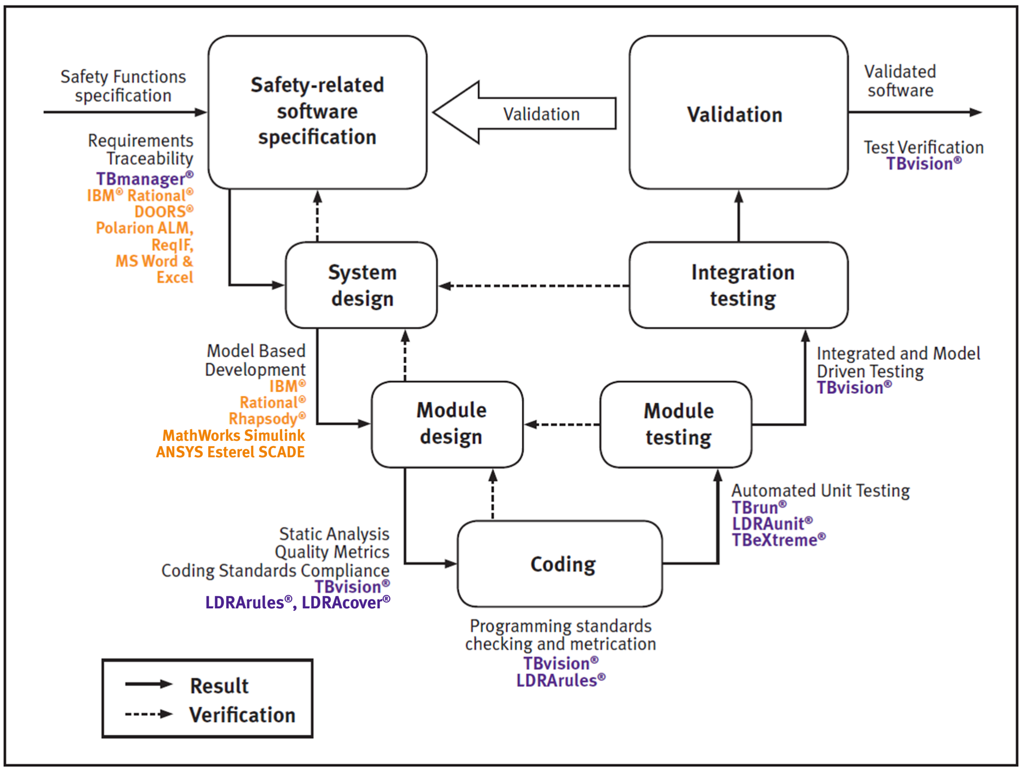
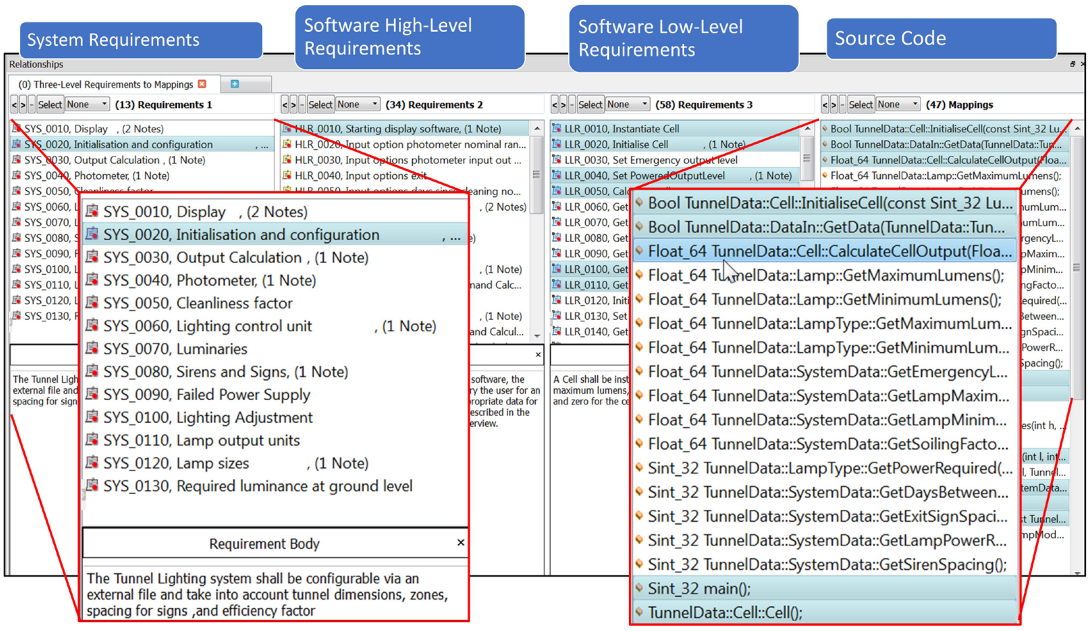


*Figure 2: Mapping the capabilities of the LDRA tool suite and complementary tools to the ISO 13849-1:2015 Simplified V-model*

## Implementing the 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 this specification to be fully implemented in the system design, for that to be fully implemented in module design, and so on.

It would be easy to dismiss the task of tracing between the development lifecycle phases as trivial. But consider, for instance, 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?



SYS_0010, Display , (2 Notes)
SYS_0020, Initialisation and configuration , ...
SYS_0030, Output Calculation , (1 Note)
SYS_0040, Photometer, (1 Note)
SYS_0050, Cleanliness factor
SYS_0060, Lighting control unit , (1 Note)
SYS_0070, Luminaries
SYS_0080, Sirens and Signs, (1 Note)
SYS_0090, Failed Power Supply
SYS_0100, Lighting Adjustment
SYS_0110, Lamp output units
SYS_0120, Lamp sizes , (1 Note)
SYS_0130, Required luminance at ground level

The Tunnel Lighting system shall be configurable via an external file and take into account tunnel dimensions, zones, spacing for signs ,and efficiency factor

HLR_0010, Starting display software, (1 Note)
HLR_0020, Input option photometer nominal ran...
HLR_0030, Input options photometer input out ...
HLR_0040, Input options exit

LLR_0010, Instantiate Cell
LLR_0020, Initialise Cell , (1 Note)
LLR_0030, Set Emergency output level
LLR_0040, Set PoweredOutputLevel , (1 Note)

Bool TunnelData::Cell::InitialiseCell(const Sint_32 Lu...
Bool TunnelData::DataIn::GetData(TunnelData::Tun...
Float_64 TunnelData::Cell::CalculateCellOutput(Floa...
Float_64 TunnelData::Lamp::GetMaximumLumens();
Float_64 TunnelData::Lamp::GetMinimumLumens();
Float_64 TunnelData::LampType::GetMaximumLum...
Float_64 TunnelData::LampType::GetMinimumLum...
Float_64 TunnelData::SystemData::GetEmergencyL...
Float_64 TunnelData::SystemData::GetLampMaxim...
Float_64 TunnelData::SystemData::GetLampMinim...
Float_64 TunnelData::SystemData::GetSoilingFacto...
Sint_32 TunnelData::LampType::GetPowerRequired(...
Sint_32 TunnelData::SystemData::GetDaysBetween...
Sint_32 TunnelData::SystemData::GetExitSignSpaci...
Sint_32 TunnelData::SystemData::GetLampPowerR...
Sint_32 TunnelData::SystemData::GetSirenSpacing();
Sint_32 main();
TunnelData::Cell::Cell();

*Figure 3: Automating Requirements Traceability with the TBmanager component of the LDRA tool suite*

The most effective way to ensure that the project is not thrown off course by such eventualities is to maintain Bidirectional Traceability of Requirements[6] to determine that all source requirements have been completely addressed, that all lower level requirements can be traced to valid source code, and that there is no source code that is surplus to requirements. 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.

A requirements traceability tool alleviates the resulting project management overhead 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 3).

## System & Module Design

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*". 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®[7], IBM® Rational® Rhapsody®[8], or ANSYS® SCADE Suite®[9] 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.

Module design can be built on the foundation of system design, and it 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*". In either case, the TBmanager component of the LDRA tool suite can help by verifying that all aspects of system design are traceable to module design, and vice versa.

## Coding

### Static Analysis

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

Both ISO 13849 ($PL_r$ c or d) and IEC 61508 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.

[6] http://westfallteam.com/Papers/Bidirectional_Requirements_Traceability.pdf Bidirectional Requirements Traceability, Linda Westfall
[7] https://uk.mathworks.com/products/simulink.html
[8] http://www-03.ibm.com/software/products/en/ratirhapfami
[9] http://www.ansys.com/products/embedded-software/ansys-scade-suite

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.  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, the enforcement of structured control flow, and the creation of data and control flow diagrams.

## Module Testing

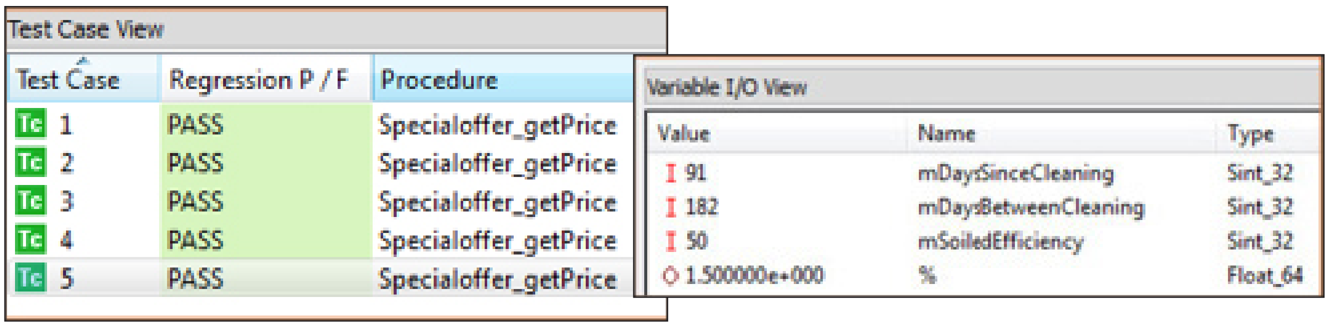
Part 2 of ISO 13849 ("*Validation*") is quite explicit in its objectives for module (or "*unit*") testing – that is, for tests which execute a subsection of the code. For example, part 2 ("*Validation*") states:

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

In this context, "*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.

Examples of how automated dynamic analysis can aid adherence to ISO 13849 objectives include "*Functional and black-box testing*", "*Automated boundary-condition test generation*", "*Fault injection tests" and "Model-driven tests*". Where the IEC 61508 development lifecycle is in use[10], more precise definitions of the type of unit tests to be implemented are provided, such as "*Test case execution from boundary value analysis*"  and "*Structural test coverage (conditions, MC/DC) 100 %*" Irrespective of whether the ISO 13849 or IEC 61508 development lifecycle is specified, extensive support is provided by the LDRA tool suite for dynamic analysis.

The TBrun component of the LDRA tool suite (Figure 4) provide a graphical user interface for unit test specification, to present a list of all defined test cases with appropriate pass/fail status. The ability to automatically create a 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. By extending that 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.



Test Case View

| Test Case | Regression P / F | Procedure |
|---|---|---|
| Tc 1 | PASS | Specialoffer_getPrice |
| Tc 2 | PASS | Specialoffer_getPrice |
| Tc 3 | PASS | Specialoffer_getPrice |
| Tc 4 | PASS | Specialoffer_getPrice |
| Tc 5 | PASS | Specialoffer_getPrice |

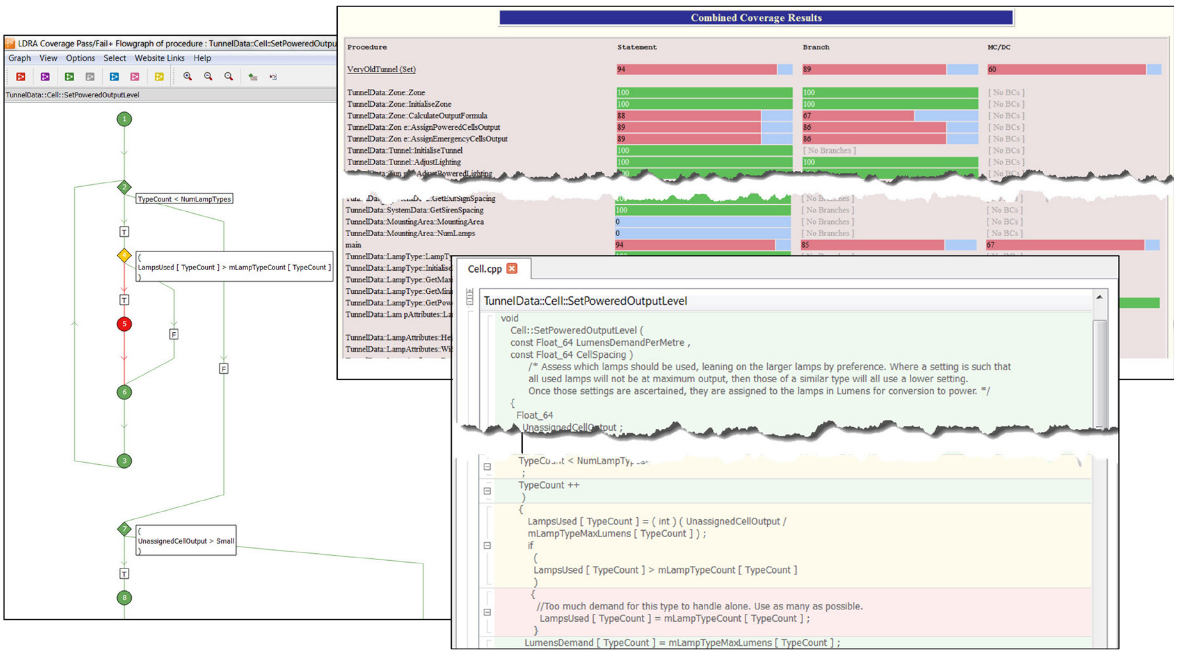
Variable I/O View

| Value | Name | Type |
|---|---|---|
| I 91 | mDaysSinceCleaning | Sint_32 |
| I 182 | mDaysBetweenCleaning | Sint_32 |
| I 50 | mSoiledEfficiency | Sint_32 |
| ◇ 1.500000e+000 | % | Float_64 |

*Figure 4: Test Case Automation with the TBrun component of the LDRA tool suite*

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 5).

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


LDRA Coverage Pass/Fail+ Flowgraph of procedure : TunnelData::Cell::SetPoweredOutpu
Graph View Options Select Website Links Help
TunnelData::Cell::SetPoweredOutputLevel
TypeCount < NumLampTypes
LampsUsed [ TypeCount ] > mLampTypeCount [ TypeCount ]
UnassignedCellOutput > Small

**Combined Coverage Results**

| Procedure | Statement | Branch | MC/DC |
|---|---|---|---|
| VeryOldTunnel (Set) | 94 | 89 | 60 |
| TunnelData::Zone::Zone | 100 | 100 | [ No BCs ] |
| TunnelData::Zone::InitialiseZone | 100 | 100 | [ No BCs ] |
| TunnelData::Zone::CalculateOutputFormula | 88 | 67 | [ No BCs ] |
| TunnelData::Zon e::AssignPoweredCellsOutput | 89 | 86 | [ No BCs ] |
| TunnelData::Zon e::AssignEmergencyCellsOutput | 89 | 86 | [ No BCs ] |
| TunnelData::Tunnel::InitialiseTunnel | 100 | [ No Branches ] | [ No BCs ] |
| TunnelData::Tunnel::AdjustLighting | 100 | 100 | [ No BCs ] |
| TunnelData::SystemData::GetSirenSpacing | 100 | [ No Branches ] | [ No BCs ] |
| TunnelData::MountingArea::MountingArea | 0 | [ No Branches ] | [ No BCs ] |
| TunnelData::MountingArea::NumLamps | 0 | [ No Branches ] | [ No BCs ] |
| main | 94 | 85 | 67 |

Cell.cpp

TunnelData::Cell::SetPoweredOutputLevel

```
void
  Cell::SetPoweredOutputLevel (
  const Float_64 LumensDemandPerMetre ,
  const Float_64 CellSpacing )
      /* Assess which lamps should be used, leaning on the larger lamps by preference. Where a setting is such that
      all used lamps will not be at maximum output, then those of a similar type will all use a lower setting.
      Once those settings are ascertained, they are assigned to the lamps in Lumens for conversion to power. */
  {
    Float_64
      UnassignedCellOutput ;
```

```
    TypeCount ++
    )
    {
      LampsUsed [ TypeCount ] = ( int ) ( UnassignedCellOutput /
      mLampTypeMaxLumens [ TypeCount ] ) ;
      if
        (
        LampsUsed [ TypeCount ] > mLampTypeCount [ TypeCount ]
        )
        {
        //Too much demand for this type to handle alone. Use as many as possible.
         LampsUsed [ TypeCount ] = mLampTypeCount [ TypeCount ] ;
        }
      LumensDemand [ TypeCount ] = mLampTypeMaxLumens [ TypeCount ] ;
```

*Figure 5: Examples of representations of structural coverage within the LDRA tool suite*

## Integration

This activity requires the integration and test of software units and/or other elements into ever larger aggregated items, ultimately verifying that the resulting integrated system behaves as intended.

Once again, for projects required to adhere to the IEC 61508 development lifecycle, the objectives are more extensive and more precisely defined[11], to include "*Dynamic analysis and testing*", "*Functional and black box testing*", and "*Model based testing*".

Extensive support is provided by the LDRA tool suite for system integration as defined by either of the specified development cycles, and 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 did 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, and again this is underpinned by dynamic structural coverage analysis as provided by both the unit test and system test facilities within the LDRA tool suite.

## Validation

Validation at the system level requires the manufacturer to confirm 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.

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.

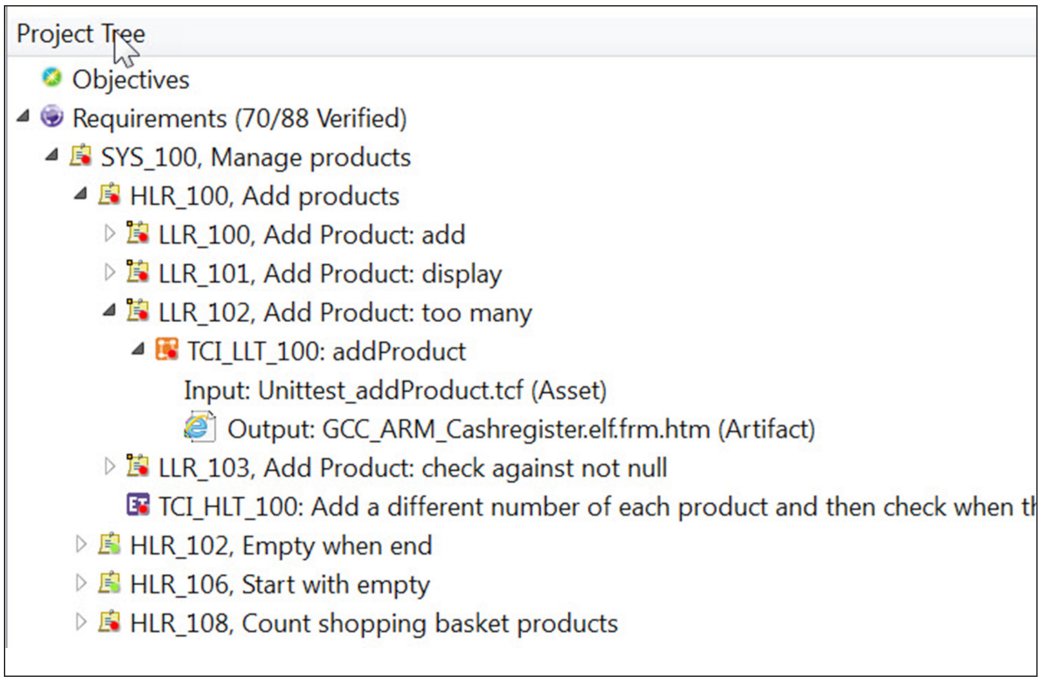
[11] IEC 61508-3 Annex A Table A.5 – Software design and development – Software module testing and integration

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. Even after product release, they present 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 and IEC 61508 development lifecycle, where the objectives are more extensive and more precisely defined[12].

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.

The TBmanager component of the LDRA tool suite helps alleviate this concern by automatically maintaining the connections between the requirements, development, and testing artefacts. activities, and highlighting where rework is required (Figure 6).



```
Project Tree
  Objectives
◢ Requirements (70/88 Verified)
  ◢ SYS_100, Manage products
    ◢ HLR_100, Add products
      ▷ LLR_100, Add Product: add
      ▷ LLR_101, Add Product: display
      ◢ LLR_102, Add Product: too many
        ◢ TCI_LLT_100: addProduct
            Input: Unittest_addProduct.tcf (Asset)
            Output: GCC_ARM_Cashregister.elf.frm.htm (Artifact)
      ▷ LLR_103, Add Product: check against not null
        TCI_HLT_100: Add a different number of each product and then check when tl
    ▷ HLR_102, Empty when end
    ▷ HLR_106, Start with empty
    ▷ HLR_108, Count shopping basket products
```

*Figure 6: Showing functions requiring retest with a red "spot" in the TBmanager component of the LDRA tool suite*

[12] *IEC 61508 Table A.8* – Software design and development –Software module testing and integration

## Conclusions

ISO 13849 is a functional safety standard that can be used across a wide range of machinery and industrial sectors. With its many sections, clauses and sub-clauses, it 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, from conception 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 even large projects with confidence in their quality.

www.ldra.com

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

***LDRA Technology***
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

ISO 9001 Quality Management

SGS TÜV SAAR FUNKTIONALE SICHERHEIT GEPRÜFT FUNCTIONAL SAFETY APPROVED

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