LDRA
Software Technology

Technical Briefing

An introduction to red-hot ASPICE compliant software development

Working with the Automotive Industry to meet the challenges of achieving cost-effective Automotive SPICE* compliance

www.ldra.com

*Software Process Improvement and Capability dEtermination

Background

There is an ever-widening range of automotive electrical and/or electronic (E/E/PE) systems such as adaptive driver assistance systems, anti-lock braking systems, steering and airbags. Their increasing levels of integration and connectivity provide almost as many challenges as their proliferation, with non-critical systems such as entertainment systems sharing the same communications infrastructure as steering, braking and control systems. The net result is a necessity for exacting development processes, from requirements specification, design, implementation, integration, verification, validation, and through to configuration.

In response to these challenges, Automotive SPICE (Software Process Improvement and Capability dEtermination) was developed 2001 by the AUTOSIG (Automotive Special Interest Group). This group consists of the SPICE User Group[1], the Procurement Forum, and automotive constructors including Audi, BMW, Daimler, Fiat, Ford, Jaguar, Land Rover, Porsche, Volkswagen, and Volvo.

Such concerns are not unique to the automotive sector, and AUTOSIG were able to draw on an earlier body of work. ISO/IEC 15504 (known as SPICE) was developed in response to the associated challenges, pre-released in 1998 as technical report, and emerged as a set of international standards in 2003/4.

The concept of the Capability Maturity Model (CMM) is central to the standard, consisting of a set of structured levels that describe how well the behaviours, practices and processes of an organization can reliably and sustainably produce required outcomes. ISO/IEC 15504 is the reference model for the maturity models against which the assessors can place the evidence that they collect during their assessment, so that the assessors can give an overall determination of the organization's capabilities for delivering products (software, systems, and IT services).

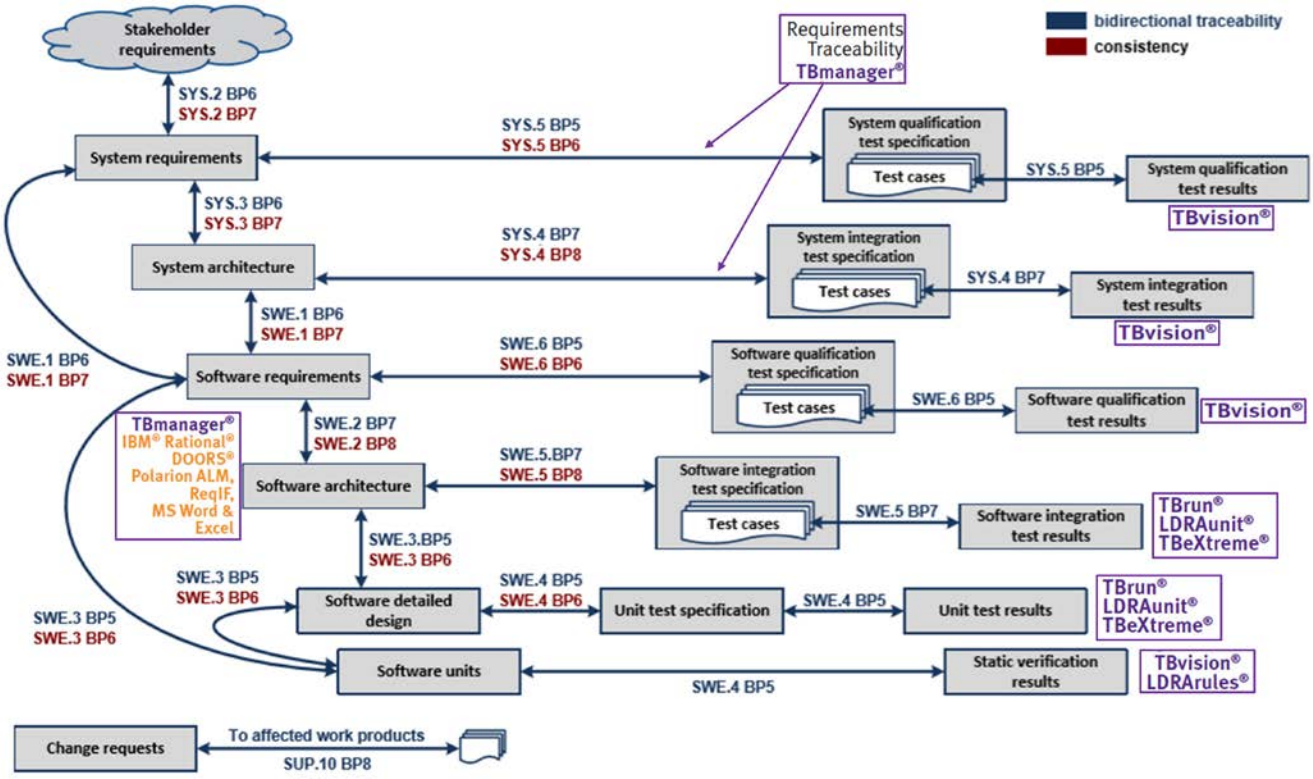

Figure 1: Mapping the capabilities of the LDRA tool suite and complementary tools to Automotive SPICE 3.1 PAM Figure D.4, "Bidirectional traceability and consistency"

The practices and processes defined within the standard align with the eight primary software verification tasks supported by the LDRA tool suite: traceability verification and process standard objective management, static analysis (design, code and quality reviews), unit testing, target testing, test verification (code coverage) and test management. Focus on all of these key areas is required to achieve an organization's software development and maintenance goals.

[1] https://www.iso.org/organization/10184.html

Figure 1 is taken from the standard and represents a lifecycle that can be summarized by reference to six software engineering processes (SWE).

SWE.1 Software Requirements Analysis

The purpose of the software requirements analysis process is to transform the software related parts of the system requirements into a set of software requirements.

The products of this phase potentially include CAD drawings, spreadsheets, textual documents and many other artefacts, and clearly a variety of tools can be involved in their production. Automating the management of the status of each of those elements and maintaining traceability between them and subsequent phases can address a project management headache (Figure 2).

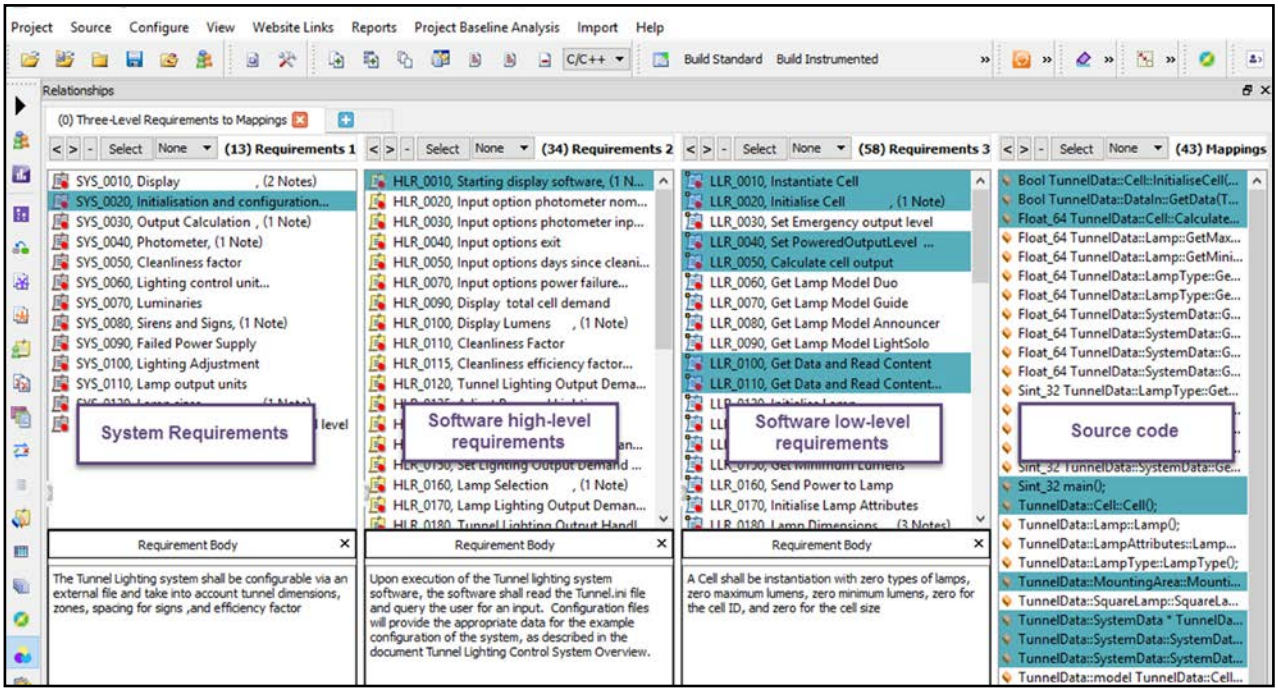

Figure 2: Traceability in the LDRA tool suite. The example detailed design is linked upstream to software safety requirements and downstream to software units.

The ideal tools for requirements management depends largely on the scale of the development, and each can be integrated with the LDRA tool suite. If there are few developers in a local office, a simple spreadsheet or Microsoft Word document may suffice. Bigger projects, perhaps with contributors in geographically diverse locations, are likely to benefit from an Application Lifecycle Management (ALM) tool such as IBM® Engineering Requirements Management DOORS® Family[2] or Siemens Polarion ALM[3].

SWE.2 Software Architectural Design

The purpose of the software architectural design process is to establish an architectural design and to identify which software requirements are to be allocated to which elements of the software, and to evaluate the software architectural design against defined criteria.

There are many tools available for the generation of the software architectural design, with graphical representation of that design an increasingly popular approach. Appropriate tools include those exemplified by IBM® Engineering Systems Design Rhapsody®[4], MathWorks Simulink[5] and Ansys SCADE Suite[6].

[2] IBM Engineering Requirements Management DOORS Family https://www.ibm.com/uk-en/products/requirements-management
[3] Siemens Polarion ALM https://polarion.plm.automation.siemens.com/
[4] IBM Engineering Systems Design Rhapsody - Developer https://www.ibm.com/uk-en/products/uml-tools
[5] MathWorks Simulink https://www.mathworks.com/products/simulink.html
[6] Ansys SCADE Suite https://www.ansys.com/en-gb/products/embedded-software/ansys-scade-suite

SWE.3 Software Detailed Design and Unit Construction

The purpose of the software detailed design and unit construction process is to provide an evaluated detailed design for the software components and to specify and to produce the software units.

The techniques suggested in the standard can be justified on the basis that they make the resulting code more reliable, less prone to error, easier to test, and/or easier to maintain. For example, language subsets such as MISRA C restrict the use of a programming language to those elements known to be least susceptible to causing problems (Figure 3).

Figure 3: Highlighting violated coding rules and guidelines in the LDRA tool suite

Bidirectional traceability between software detailed design and software units is a key requirement of the standard. Automating that traceability reduces both management overhead and the potential for error, particularly when unanticipated changes arise. In such circumstances, impact analysis reports help to quantify the overhead associated with such changes and ensure that they are implemented in full (Figure 4).

TunnelData::Lamp::GetMaximumLumens	Lamp.cpp	LLR_0140	Software Detailed Design
TunnelData::LampAttributes::Height	Lampmodel.cpp	LLR_0180	Software Detailed Design
TunnelData::LampAttributes::LampAttributes	Lampmodel.cpp	LLR_0170	Software Detailed Design
TunnelData::LampAttributes::Width	Lampmodel.cpp	LLR_0190	Software Detailed Design
TunnelData::LampAttributes::Drain	Lampmodel.cpp	LLR_0210	Software Detailed Design

Figure 4: Impact analysis report generated by the LDRA tool suite

SWE.4 Software Unit Verification

The purpose of the software unit verification process is to verify software units to provide evidence for compliance of the software units with the software detailed design and with the non-functional software requirements.

Each developed software unit needs to be tested with reference to the software detailed design. Test procedures then need to be authored, reviewed, and executed to ensure the software unit does not contain any undesired functionality. Unit tests can then be executed on the target hardware and/or simulated environment. Once the test procedures are executed, actual outputs are captured and compared with the expected results. Pass/Fail results are then reported and requirements are verified accordingly.

The LDRA tool suite automates the unit test process by exposing the software interface at the function scope allowing the user to enter inputs and expected outputs. The tool suite then generates a test harness, which is compiled and executed on the target hardware. Actual outputs are captured, along with structural coverage data, and then compared with the expected outputs specified in the test cases (Figure 5).

Should changes become necessary – perhaps as a result of a failed test, or in response to a requirement change - then all impacted tests would need to be re-run (regression tested). These regression tests can be automated and systematically re-applied as development progresses to ensure that new functionality does not compromise any that is established and proven.

Bidirectional traceability between software detailed design and the unit test specification is a key requirement of the standard. Automated traceability and the provision of associated reporting reduces both management overhead and the potential for error, particularly when unanticipated changes arise (Figure 6).

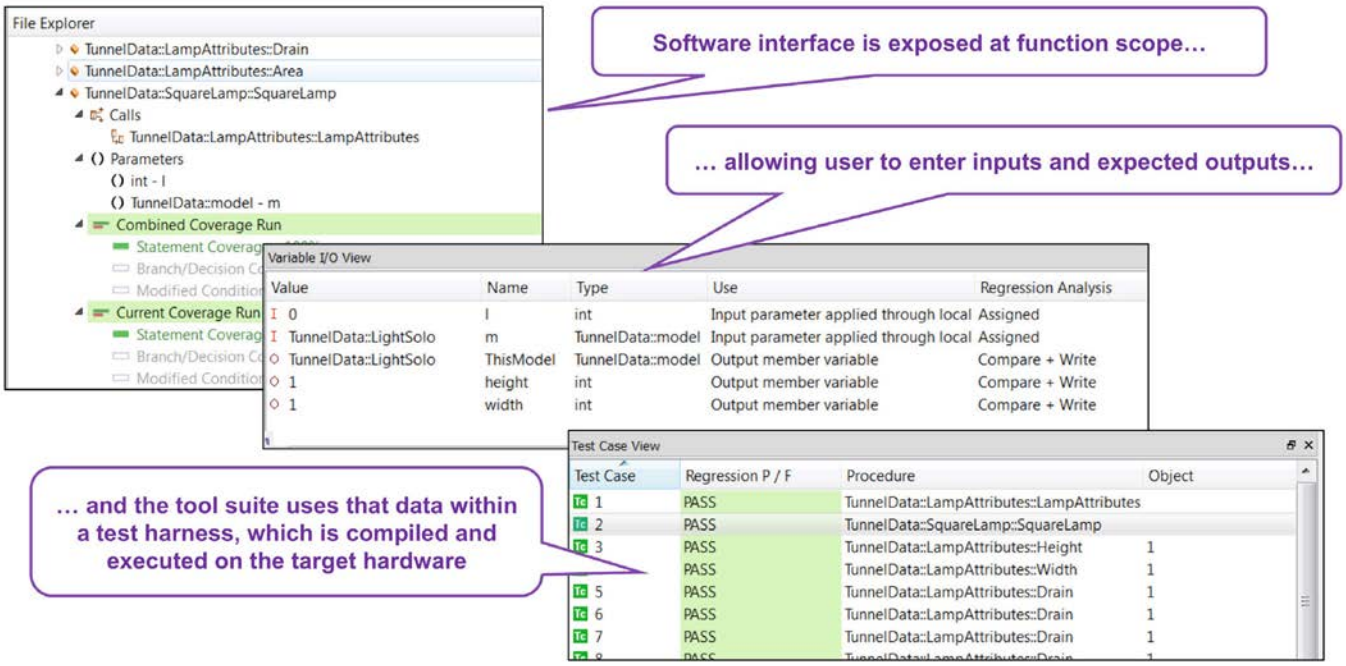

Figure 5: Automation of unit test using the LDRA tool suite

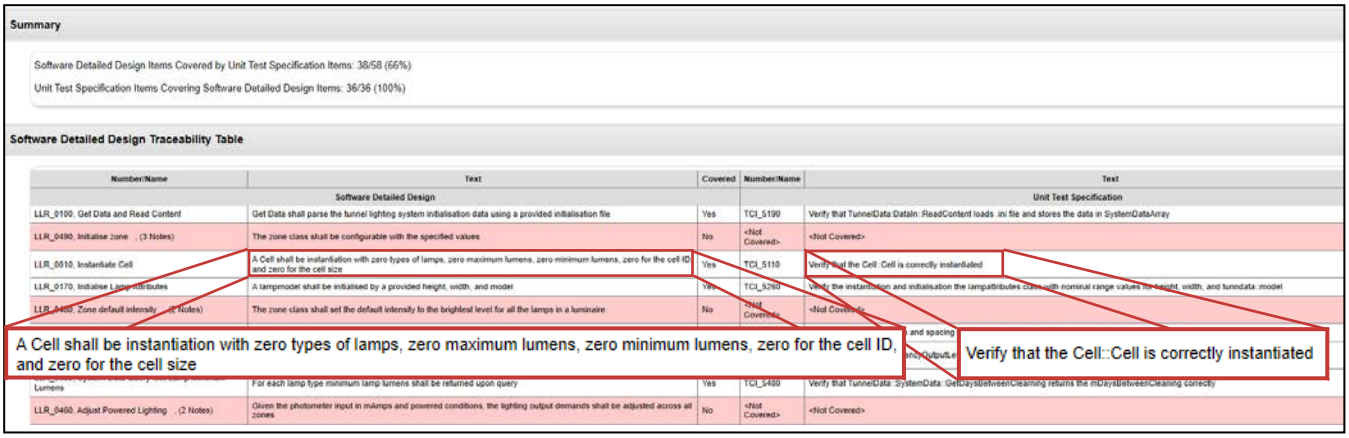

Figure 6: Detailed traceability report generated by the LDRA tool suite

Software test and model based development

These static and dynamic analyses can be integrated with several different model based development tools, such as IBM Engineering Systems Design Rhapsody, MathWorks Simulink, and Ansys SCADE Suite. 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.

Model based development offers many advantages to developers of automotive software and many modelling tools include integrated model and auto-generated code testing features. However, an automated approach to testing that is integrated with the modelling tool and yet independent from it helps to offset concerns relating to systemic faults.

Figure 7 illustrates one example of how an integration with IBM Engineering Systems Design Rhapsody can be deployed using an approach appropriate for use with “back-to-back” testing. Design models are developed with Rhapsody and verified with Rhapsody Test Conductor. Then, code is generated from Rhapsody, instrumented by the LDRA tool suite, and executed in Software In the Loop (SIL or host), or Processor In the Loop (PIL or target) mode. Structural coverage is then collected and structural coverage reports can be generated at the source code level.

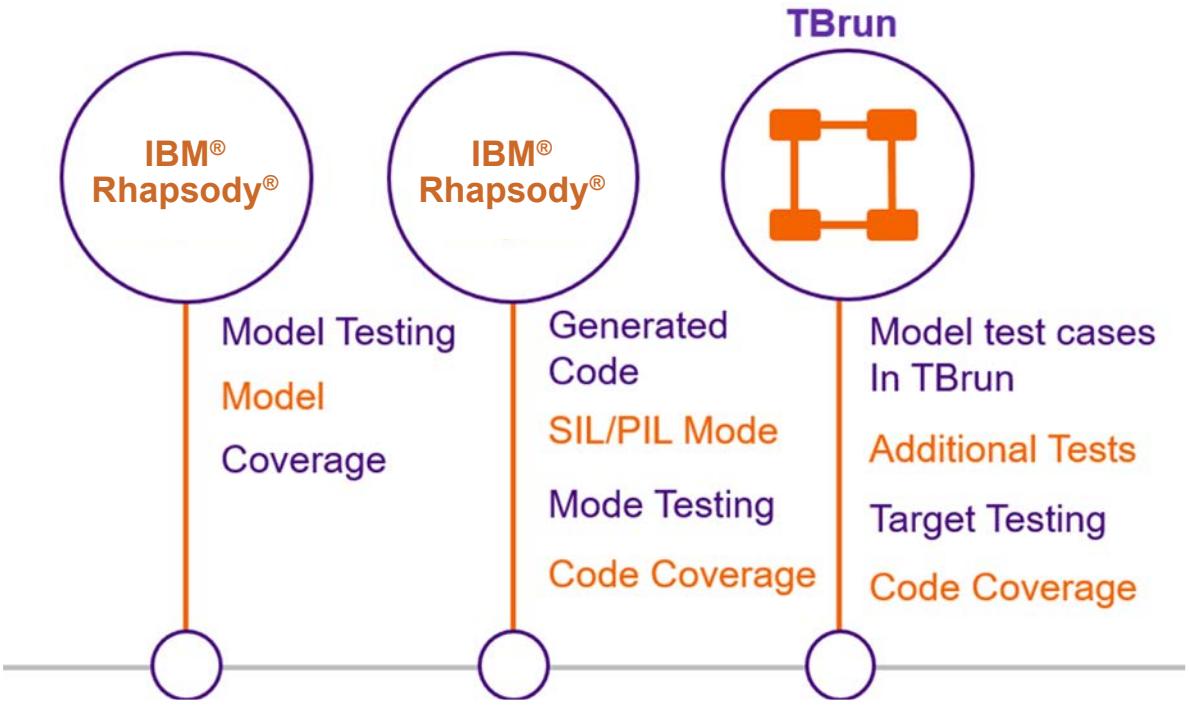

Figure 7: Generating structural coverage data of auto generated code with IBM Engineering Systems Design Rhapsody and the LDRA tool suite

The generated source code can be analysed statically to ensure compliance with an appropriate coding standard, such as MISRA C:2012 Appendix E[7]. Additional dynamic testing can be performed at the source level from within the LDRA tool suite. Requirements based tests can be created to verify functionality and collate structural coverage. Test data can also be imported from Simulink and used to migrate test data to the LDRA tool suite for efficiency.

Real time embedded systems based on auto generated code usually also include some level of conventionally written code. Software for board support packages, interrupt handlers, drivers, and other lower-level code is typically hand coded. Legacy code is almost always part of deployed systems. These portions of the system can be verified using traditional methods using the LDRA tool suite alongside auto-generated code.

SWE.5 Software Integration and Integration Test

The purpose of the software integration and integration test process is to integrate the software units into larger software items up to a complete integrated software consistent with the software architectural design and to ensure that the software items are tested to provide evidence for compliance of the integrated software items with the software architectural design, including the interfaces between the software units and between the software items.

LDRA static analysis tools contribute to the verification of the design by means of the control and data flow analysis of the code derived from it, providing graphical representations of the relationship between code components for comparison with the intended design (Figure 8). A similar approach can also be used to generate a graphical representation of legacy system code, providing a path for additions to it to be designed and proven in accordance with ASPICE principles.

[7] https://www.misra.org.uk/tabid/72/Default.aspx

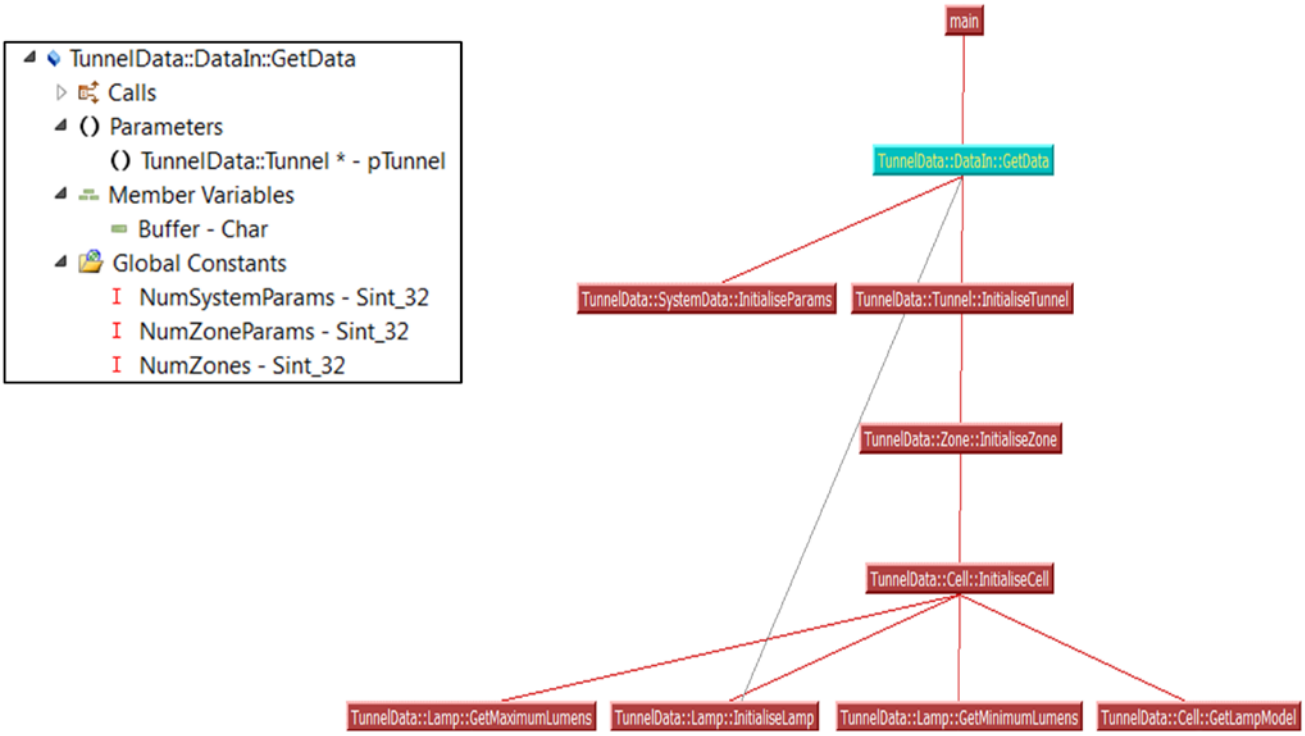

Figure 8: Diagrammatic representations of control and data flow generated from source code by the LDRA tool suite aid verification of software architectural design

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. It is desirable for all dynamic testing to use environments that correspond closely to the target environment and hence test dependencies between hardware and software.

Within the LDRA tool suite, unit tests become integration tests as units are tested as part of a call tree, rather than in isolation. Exactly the same test data can be used to validate the code in both cases.

The inputs and expected outputs defined in the test cases are typically derived from requirements to ensure intended functionality is verified. Various other forms of tests including negative tests, fault injection and robustness tests are available using the same mechanism.

The analysis of boundary values can be automated using an “extreme test” facility to automatically generate a series of unit test cases. The same facility also provides a facility for the definition of equivalence boundary values such as minimum value, value below lower partition value, lower partition value, upper partition value and value above upper partition boundary. Features like automated stub management, global variable declarations, and exception handling help to complete a comprehensive unit test facility.

SWE.6 Software Qualification Test

The purpose of the Software Qualification Test Process is to ensure that the integrated software is tested to provide evidence for compliance with the software requirements.

The sequential nature of the process defined by ASPICE can seem simplistic at first sight. In theory, if the V-model is adhered to, then the requirements will never change, each phase will follow in sequence, and tests will never throw up a problem.

Consider what happens if there is a code change in response to a failed integration test, or where there is a change in a customer requirement. Such scenarios can quickly lead to situations where the traceability between the products of software development falls down if the integrity of each development phase and the artefacts generated by them are not maintained. A sequence of similar issues can ultimately lead to a situation where the completed project does not fulfil functional, functional safety, or cybersecurity requirements.

For that reason, ASPICE incorporates the principle of bidirectional traceability, requiring ongoing maintenance and integrity of the artefacts generated by each phase of the development lifecycle. Adherence to this principle ensures not only that the delivered system accurately reflects the requirements of the stakeholders as confirmed during the software qualification test phase, but also confirms that there is no superfluous code within the system – important from both a safety and security perspective.

The double headed arrows in Figure 1 represent bidirectional traceability, confirming that each software "production" phase on the right side of the V accurately and completely reflects the "design" phase that specified it on the left. Figure 2 illustrates its automation.

Base practices, process attributes, and functional safety

Each of these SWEs is broken down in the standard to a number of base practices (BPs). For example, the base practices associated with software unit verification are shown in Figure 9.

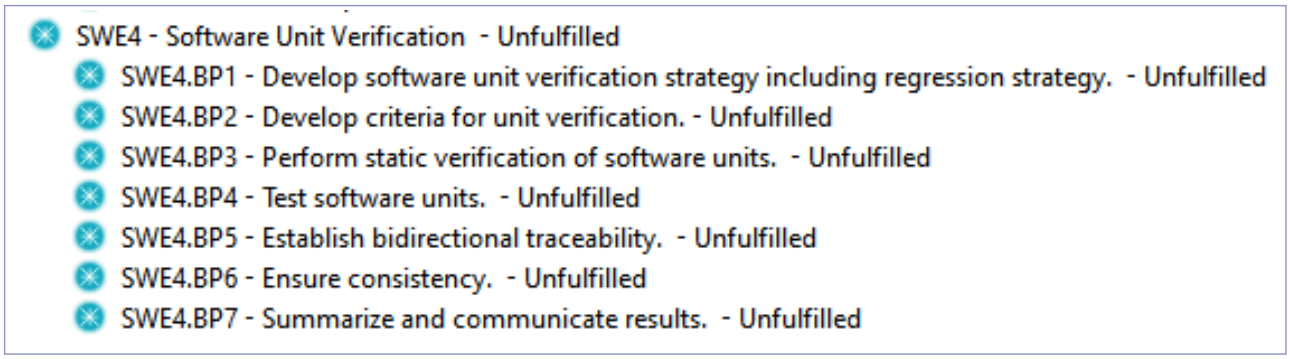

Figure 9: Base practices associated with software unit verification, as represented in the LDRA tool suite

Clearly the level of thoroughness and expertise applied to each of these activities can vary enormously, and with it the level of quality assurance. Take, as a further example, the testing of software units (BP4). The standard expands that a little "*Test software units using the unit test specification according to the software unit verification strategy. Record the test results and logs.*"

The standard provides for the assessment of capability levels associated with each of these base processes (Figure 10).

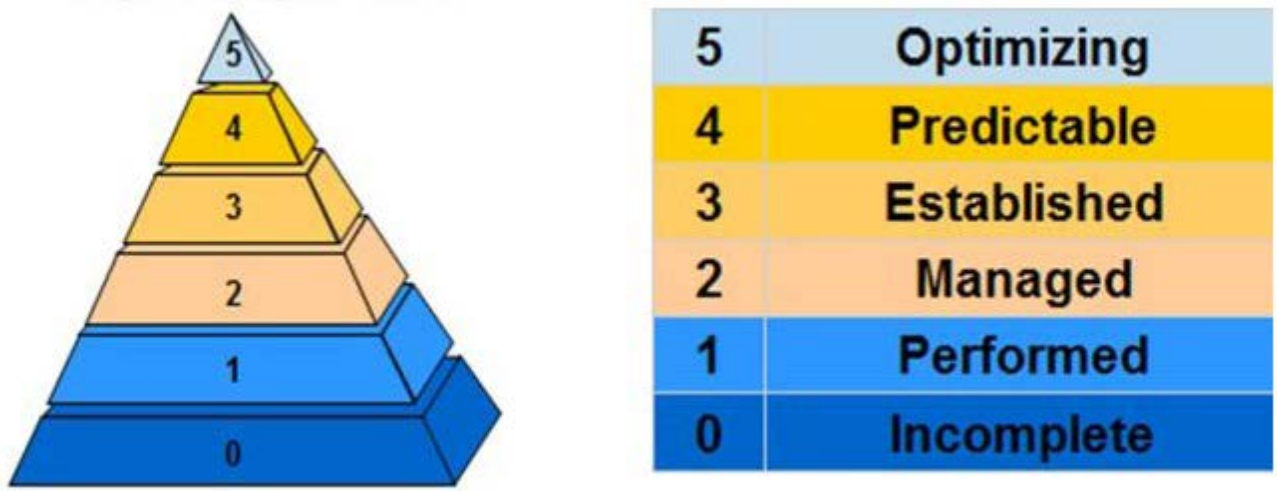

Figure 10: Capability levels in ASPICE

Any assessment of compliance will look to confirm the completeness of that test specification and the effectiveness of adherence to it in order to assess an appropriate capability level, but outside that remit there are decisions to be made relating to the rigour applied to each activity.

The thoroughness of unit test, for example, can vary considerably.

- At its most rudimentary level, unit test can consist of the superficial confirmation that each software function fulfils its functional specification.
- A more thorough test would include evidence of code coverage, showing that all code relates to a requirement and that all requirements are fulfilled by code.
- Beyond that, the robustness of the code might be exercised to show that boundary and out of range values are handled adequately by the code.
- Ultimately, object code verification might be deployed to provide evidence that the object code, like the source code, is also shown to be both pertinent and thoroughly exercised.

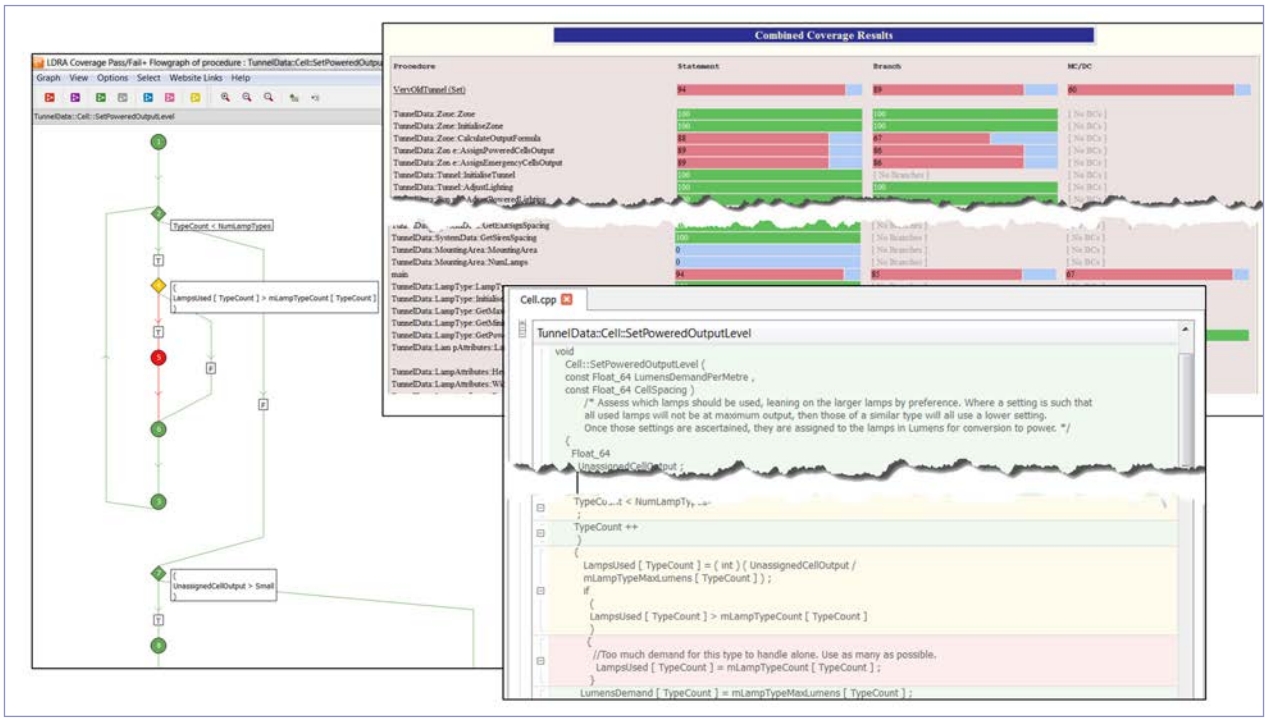

Figure 11: Examples of representations of function and call coverage within the LDRA tool suite

Clearly, the more demanding tests would likely apply to a braking system and only the lesser ones to in-car entertainment. Functional safety issues of this nature are outside the remit of Automotive Spice, but within the scope of the standard ISO 26262 Second Edition “Road vehicles — Functional safety” which might be considered complementary to it.

In summary

ASPICE represents best practice in the development of functionally safe automotive software. A key concept for the standard Capability Maturity Model (CMM) - a set of structured levels that describe how well the behaviours, practices and processes of an organization can reliably and sustainably produce required outcomes.

These practices and processes can be represented by a V-model, enhanced by bidirectional traceability to ensure that each phase of development always accurately reflects the one before it. The development and validation processes required for each phase are broken down by the standard, but there is no provision in ASPICE for the variation in thoroughness proportional to the functional safety demanded of an application. Leveraging ISO 26262 in tandem with ASPICE would address that issue.

www.ldra.com

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, 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