LDRA
Software Technology

Technical White Paper

Verification of airborne software in compliance with DO-178C

www.ldra.com

Contents

In response to the increased use of software in airborne systems, the Radio Technical Commission for Aeronautics organization [1] (now known as RTCA, Inc.) in collaboration with EUROCAE [2], published the 1982 guidance document "Software Considerations in Airborne Systems and Equipment Certification". The document, referenced as RTCA DO 178 [3] and EUROSAE ED-12 [4] respectively by the two authorities, came to be accepted as the international certification standard for airborne software. It was updated and rewritten in 1992 as DO 178B/ED 12B [5] [6].

Significantly extended in 2011 as DO 178C/ED-12C [7] [8] to meet today's aviation industry needs and those of the aviation regulators that have adopted it (including, FAA [9], EASA [10], TCCA [11] and many more) the standard was been joined by a series of supplements and guidance documents (DO-330 [12], DO-331 [13], DO-332 [14], DO-333 [15]) that further extended its scope to cover newly emerged needs and technologies. LDRA has participated extensively on both the DO-178B and DO-178C committees over nearly two decades. Mike Hennell, LDRA's CEO, was instrumental in the inclusion of several test measurement objectives in the standard, including those relating to structural coverage analysis. The LDRA tool suite [16] was a forerunner in automated verification for certification to both the DO-178B standard for airborne software systems, and to its companion standard, DO-278/ED-109 [17] [18] for ground-based systems.

Since 2020, ongoing work around the development of DO-178C has focused on maintaining the safety and reliability of airborne software systems in the wake of further technological advances. Examples of resulting guidance include "AC 20-193, Multi-core Processors" [19] and "AC 20-170A, Integrated Modular Avionics" [20].

Referencing DO-178C

It is useful to understand the relationship between the body of DO-178C and the tables in Annex A.

The main body of DO-178C provides detailed guidance on the processes and objectives required for developing and certifying airborne software. It covers various aspects such as planning, development, verification, configuration management, and quality assurance. Each section outlines specific activities and requirements that must be met to ensure the software's safety and reliability.

The tables in DO-178C Annex A serve as a practical tool to summarize and clarify the objectives and activities described in the main body. These tables provide a structured way to ensure that all necessary steps are followed and documented – but they do not include the same level of detail.

For that reason, this white paper references the sections of the main body only. For completeness, Appendix A and Appendix B of this paper reproduce pertinent extracts from DO-178C tables.

DO-178C in context

Figure 1 shows the current version of DO-178C and sister document DO-278A [21] in the context of the broader avionics development and certification framework.

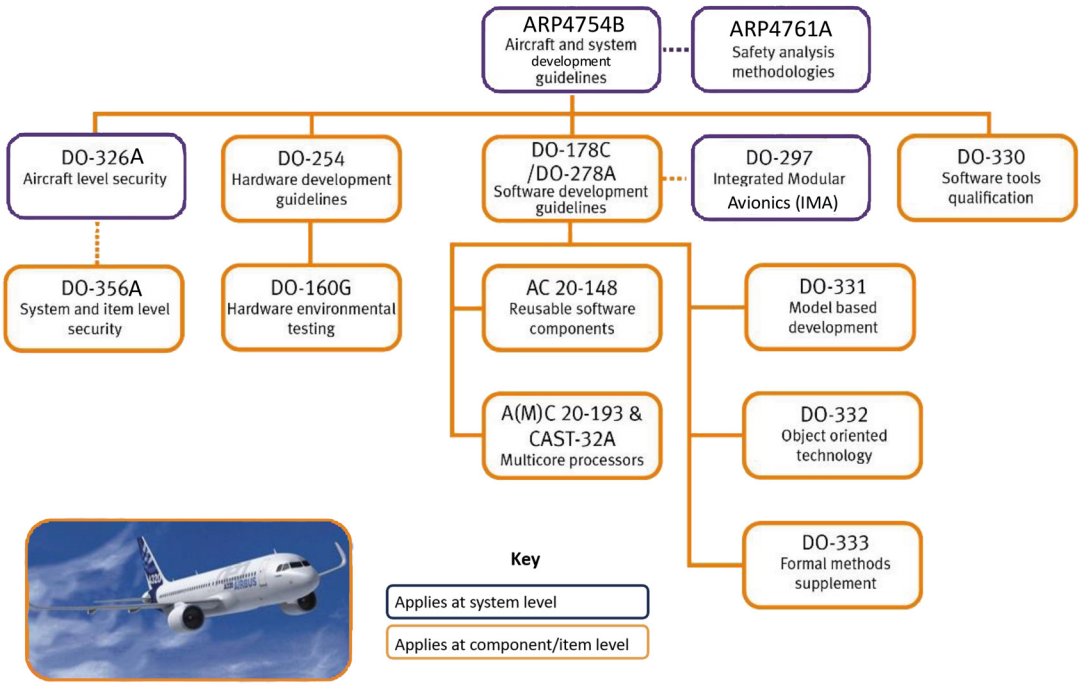


Figure 1: DO-178C in the context of the broader avionics development and certification framework

ARP4754B [22] provides the overarching process for system development and certification. Its sister document, ARP4761A [23], describes methodologies for safety assessment.

These methodologies inform and integrate with the system development processes of ARP4754B, the software development guidelines of DO-178C, and the hardware development guidelines of DO-254/ED-80 [24] [25] & DO-160G/ED-14G [26] [27].

Many modern aircraft use Integrated Modular Avionics (IMA) architectures. These provide a flexible and efficient way to integrate multiple avionics functions and applications on a common computing platform. DO-297/ED-124 [28] [29] provides development guidance for the implementation of IMA architectures. It complements DO-178C processes which are equally relevant whether they are used within an IMA environment, or not.

As challenges emerge and technologies advance, so standards are introduced or adapted to accommodate them (Figure 2).

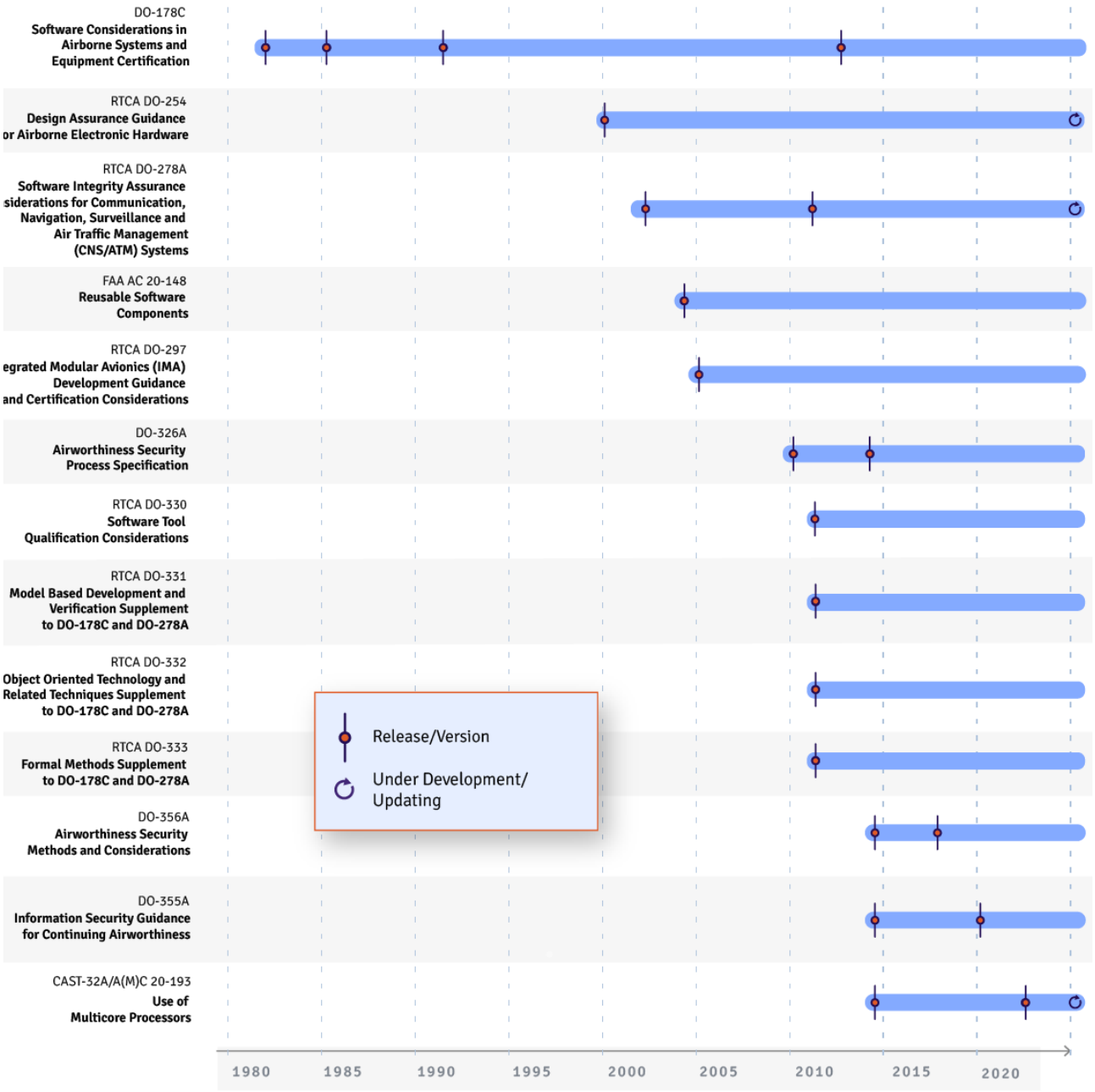


Figure 2: The avionics certification framework is continuously evolving

The emergence of connectivity in aircraft serves to illustrate that continuous improvement. Connected services enhance operational efficiency, passenger experience, safety, and maintenance practices through seamless integration of technology and data exchange capabilities. However, connectivity brings an inevitable associated disadvantage of the possibility of compromise by bad actors which is an obvious concern in such a safety-critical environment.

Emerging security-specific concerns precipitated the development and 2014 publication of DO-326A/ED-202A [30] [31] which was supplemented by DO-356A/ED-203A [32] [33] in 2018. DO-326A focuses on the security aspects of aircraft and aircraft systems, ensuring the security of onboard systems against potential cyber threats. DO-356A provides guidelines for the assurance aspects of aircraft cybersecurity, ensuring that security controls are properly implemented.

Supplementary guidance documents keep DO-178C and DO-278A within manageable proportions and allow complementary topics to be examined in more detail. They also allow emerging technologies to be addressed without requiring a revision of the two core documents. Many of the supplements introduced here are referenced in context later in this white paper:

- **DO-330/ED-215** [12] [34] is concerned with tool qualification - a generic term to describe a process designed to ensure that the risk of a tool error impacting the safety of a system is acceptably low. DO- 330 provides guidance in the achievement of tool qualification for tools to be used in the pursuit of compliance with DO-178C & DO-278C. It is also designed for use in other domains unlike the other supplementary documents DO-331, DO-332, and DO-333.
- **DO-331/ED-216** [13] [35] details additional objectives that apply when using model-based development. It also provides advice on how the objectives in DO-178C & DO-278A should be approached in that environment.
- **DO-332/ED-217** [14] [36] includes additional objectives that apply when using object-oriented programming and complementary practices. It also provides advice on how the objectives in DO-178C & DO-278A should be approached in that environment.
- **DO-333/ED-218** [15] [37] identifies additional objectives that apply when using formal methods as part of a software life cycle in the context of DO-178C & DO-278A objectives.
- **AC 20-193/AMC 20-193** [38] [19] (collectively **A(M)C 20-193**) describes a set of objectives to be fulfilled when multicore processors are used in a project that is compliant with DO- 178C or DO-278A.
- **AC 20-148** [20] addresses the certification of software intended for reuse across multiple systems. AC 20-148 is not feasible in the EASA context because "it makes use of acceptance letters for software parts." [39]
- **AC 20-170A/AMC 20-170A** (collectively **A(M)C 20-170A**) provides guidance on the development, verification, validation, integration, and approval of Integrated Modular Avionics (IMA) systems. These often include software components that must comply with DO-178C standards.

DO-178C applications

DO-178C is applied to a wide range of critical software applications on aircraft, ensuring their safety and reliability. These applications include Line-Replaceable Units (LRUs) – that is, modular components that can be quickly swapped out relatively easily. Examples include flight control systems, which manage the aircraft's stability and manoeuvrability; navigation systems, which provide essential data for route planning and positioning; communication systems, which facilitate vital exchanges between the aircraft and ground control; and surveillance systems, which monitor the aircraft's environment and detect potential hazards.

Supplementary guidance is integrated as necessary. For example, developers using Model-Based Development (MBD) to develop an ice protection system would call upon DO-331 to provide the guidance to apply DO-178C to their chosen development toolchain. Similarly, system developers leveraging multicore processors would look to A(M)C 20-193 to understand how their development life cycle should be adapted to suit.

ARP4754B and DALs

ARP4754B requires that prior to system development, functional hazard analyses and system safety assessments are performed to determine the contribution of the system to potential failure conditions. The severity of failure conditions on the aircraft and its occupants are then used to determine a Design Assurance Level (DAL), as shown in Figure 3.

DAL	Failure Condition	Description
A	Catastrophic	Failure Conditions, which would result in multiple fatalities, usually with the loss of the airplane
B	Hazardous	Failure Conditions, which would reduce the capability of the airplane or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be: • A large reduction in safety margins or functional capabilities; • Physical distress or excessive workload such that the flight crew cannot be relied upon to perform their tasks accurately or completely, or • Serious or fatal injury to a relatively small number of the occupants other than the flight crew.
C	Major	Failure Conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to the flight crew, or physical distress to passengers or cabin crew, possibly including injuries.
D	Minor	Failure Conditions which would not significantly reduce airplane safety, and which involve crew actions that are well within their capabilities. Minor Failure Conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as routine flight plan changes, or some physical discomfort to passengers or cabin crew.
E	No effect	Failure Conditions that would have no effect on safety; for example, Failure Conditions that would not affect the operational capability of the airplane or increase crew workload.

Figure 3: Design Assurance Levels as described in DO-178C (Table 2-1)[1].

The ARP 4754B development process then allocates the associated DALs to the subsystems that implement the system's electronic hardware and software requirements. DO-178C establishes five "software levels" and modulates objectives that must be satisfied. This means that the effort and expense of producing a system critical to the continued safe operation of an aircraft (e.g. a flight control system) is necessarily higher than that required to produce a system with only a minor impact on the aircraft in the case of a failure (e.g. a bathroom smoke detector).

It also covers the complete software life cycle: planning, development and integral processes to ensure correctness and robustness in the software. The integral processes include software verification, software quality assurance, configuration management assurance and certification liaison with the regulatory authorities.

The standard does not oblige developers to use analysis, test, and traceability tools in their work. However, these tools improve efficiency in all but the most trivial projects to the extent that they have a significant part to play in the achievement of the airworthiness objectives for airborne software throughout the development life cycle. Specialised tools exemplified by the LDRA tool suite are used to help achieve DO-178C objectives including bi-directional traceability, test management, source code static analysis, and dynamic analysis of both source and object code. LDRA tools also automate the management of the many artifacts generated by these activities and which need to be systematically collated to demonstrate compliance.

[1] Based on table 2-1 from RTCA DO-178C, Copyright © 2011 RTCA, Inc. All rights acknowledged.

DO-178C Process Objectives

Figure 4: DO-178C process objectives timeline.

DO-178C recognizes that software safety must be addressed systematically throughout the software life cycle. This involves life cycle traceability, software design, coding, validation and verification processes used to ensure correctness, control and confidence in the software. Several mechanisms are defined to help ensure that the processes are adhered to, and to provide evidence of that adherence.

Planning documents

Collectively, the DO-178C planning documents provide a comprehensive framework for the development and certification of airborne software (Figure 4). They are intended to ensure that all activities are planned, controlled, and verified according to requirements:

- **Plan for Software Aspects of Certification (PSAC):** The PSAC outlines how the software development process will comply with DO-178C requirements. It details the life cycle, processes, standards, tools and techniques that will be leveraged, identifies key roles and responsibilities, and defines the objectives to be met at each stage.
- **Software Development Plan (SDP):** The SDP provides a detailed roadmap for software development activities. It covers the methodologies and standards to be followed, the development environment, configuration management practices, and the schedule for software development tasks. It also outlines how traceability will be maintained between requirements, design, code, and tests.
- **Software Verification Plan (SVP):** The SVP details how the software verification activities will be conducted to ensure compliance with DO-178C. It includes the strategies and techniques, the types of reviews and tests to be performed, and the criteria for success. It also specifies the roles and responsibilities of verification personnel and the tools to be used.
- **Software Configuration Management Plan (SCMP):** The SCMP ensures that software configuration management practices are in place to control changes and maintain the integrity of the software products. It also defines personnel roles and responsibilities.
- **Software Quality Assurance Plan (SQAP):** The SQAP ensures that quality assurance activities are integrated into the software development process. It outlines the quality assurance processes and activities, criteria for acceptance, reporting mechanisms. It also details the roles and responsibilities of quality assurance personnel.
- **Tool Qualification Plan (TQP):** The TQP details the qualification process for tools used in the development and verification of the software. The aim is to ensure that tools do not introduce or fail to eliminate errors within the development environment, and to provide evidence of that.

Stages Of Involvement (SOI)

There are four Stage of Involvement (SOI) reviews specified by DO-178C (Figure 4). These are designed to ensure that software development processes and outputs adhere to the document's stringent requirements

FAA Designated Engineering Representatives (DERs) and EASA Subject Matter Experts (SMEs) police the DO-178C process particularly through their involvement with SOI audits. They provide oversight and approval at each stage, ensuring that the software development processes adhere to DO-178C guidelines. They review key documents such as the Plan for Software Aspects of Certification (PSAC), Software Configuration Index (SCI), and Software Accomplishment Summary (SAS), and they verify that the software meets all safety and performance requirements.

The four SOIs are as follows:

- **SOI#1** - Planning Review: This initial stage involves a thorough review of the planning documents. It looks to validate the groundwork for the software development life cycle, ensure that all planned activities meet regulatory expectations, and confirm that they set a solid foundation for the development phases.
- **SOI#2** - Development Review: At this stage, the focus shifts to examining the software development process which includes requirements, design, and coding activities. The review is designed to ensure that these activities are aligned with the approved plans and that they meet the necessary quality and compliance standards defined in DO-178C.
- **SOI#3** - Verification Review: The third review is designed to ensure that the software complies with its specified requirements and that the verification processes are rigorously followed. It aims to confirm that the software performs as intended and that all verification procedures, including testing and analysis, are thoroughly documented and executed according to the standards.
- **SOI#4** - Certification Review: This final stage involves a comprehensive review of all compliance evidence collected throughout the software development life cycle, ensuring its completeness and correctness prior to submission for certification.

Each of these stages plays a vital role in demonstrating that the software meets the stringent safety standards required for certification. The SOI reviews provide structured checkpoints at various phases of the software development life cycle, ensuring a disciplined and methodical approach to achieving compliance with DO-178C.

Software Accomplishment Summary (SAS)

The SAS document is a comprehensive summary that encapsulates all the steps taken during the DO-178C software development and verification processes. It includes a recap of the software development activities, a summary of the verification results, and findings from the qualification of verification tools (Figure 4).

The SAS is reviewed during the final Stage of Involvement (SOI #4) by the relevant certification authorities to ensure that all aspects of the software development and verification are complete and satisfactory. It therefore serves as a final report that demonstrates the software has met all the necessary DO-178C objectives and is ready for certification.

Evidential artifacts presented in a comprehensive, clear, and concise manner as part of the SAS are a key part of any successful compliant project. It is, of course, important to complete the appropriate validation and verification exercises, but it is equally important that the authorities can verify their successful completion (Figure 5).

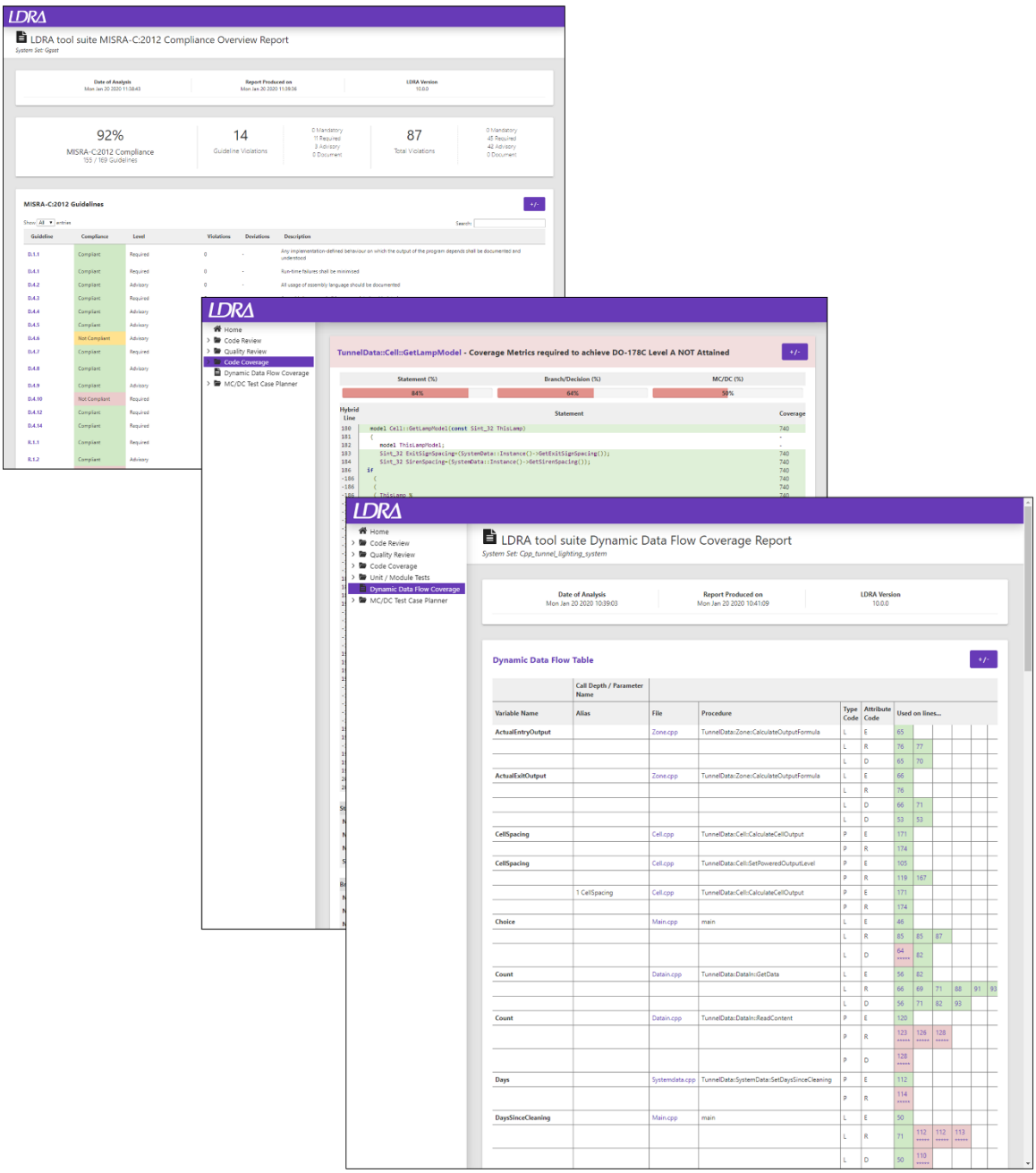


Figure 5: Reporting with the LDRA tool suite.

Compliance management

Project management problems are common to the development of many safety-critical software projects. One significant issue is managing and maintaining traceability of requirements throughout the software development life cycle. Ensuring that all requirements are accurately captured, implemented, and linked to corresponding code and test cases can be challenging, leading to potential gaps and inconsistencies that complicate compliance efforts.

Reliance on manual processes often results in inconsistencies and errors, which are particularly problematic in safety-critical environments where stringent adherence to processes is crucial. Configuration management poses another challenge, as handling different versions of software artifacts and maintaining the integrity of configurations throughout the project life cycle is complex.

The creation of the extensive documentation required for each SOI can be labour-intensive and prone to errors, which can hinder the certification process. Finally, many projects suffer from a lack of visibility into the project status, making it difficult for managers to track progress, identify issues early, and make informed decisions.

LDRA Certification Services (LCS) [40] offers the LDRA Compliance Management System (LCMS) [41] to address these challenges. LCMS is a cloud-based or locally hosted packaged solution which includes comprehensive compliance documentation, process and document review tools, together with Level A FAA Designated Engineering Representative (DER) and EASA Subject Matter Expert (SME) support. It integrates with the LDRA Tool Suite to provide a comprehensive solution for developing and certifying safety-critical software (Figure 6).

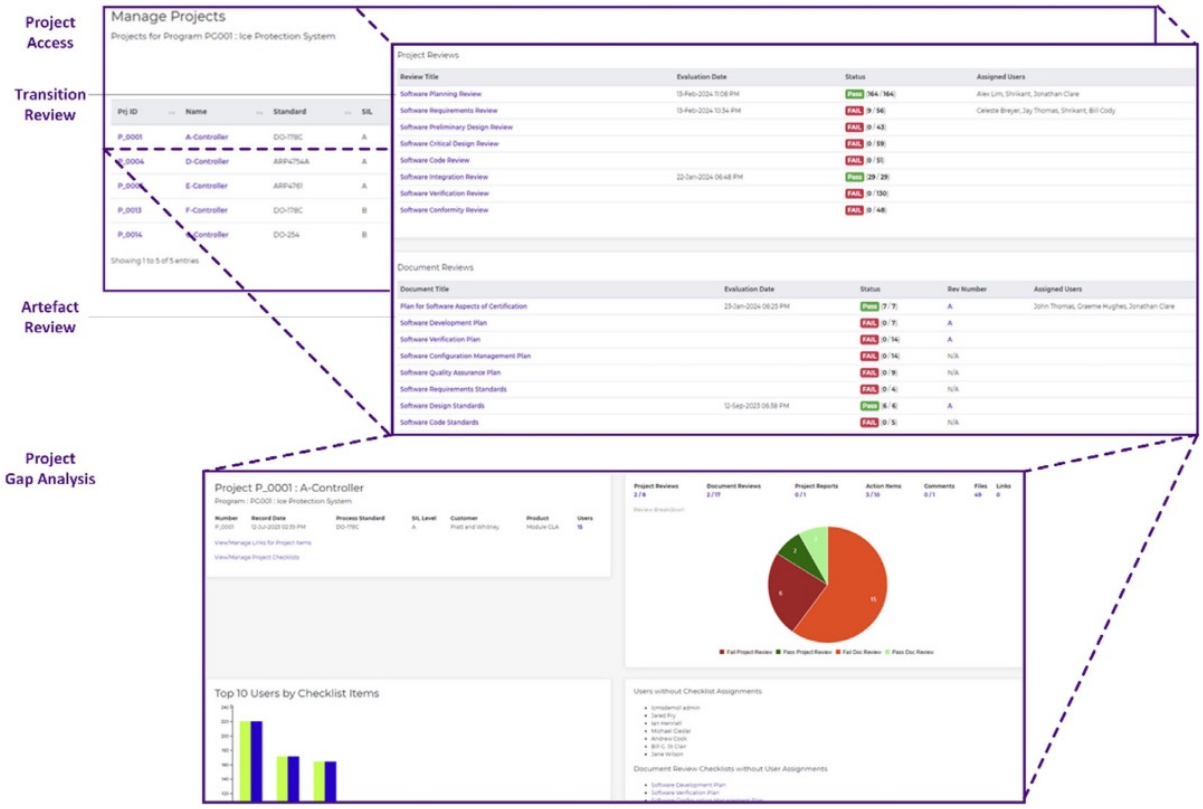


Figure 6: LCS LCMS integrates with the LDRA tool suite to provides a comprehensive solution for the development of compliant software

DO-178C Software development and verification processes

The DO-178C software life cycle and the activities it specifies fit within the framework defined by the standard's life cycle processes.

Key elements of the DO-178C software life cycle include the practices of traceability and structural coverage analysis. Bi-directional traceability must be established across the life cycle, from system requirements to software high-level requirements, and from software high-level requirements to low-level requirements. Low-level requirements must then be linked to the source code in which they are implemented, through to test cases, test procedures and test results (Figure 7).

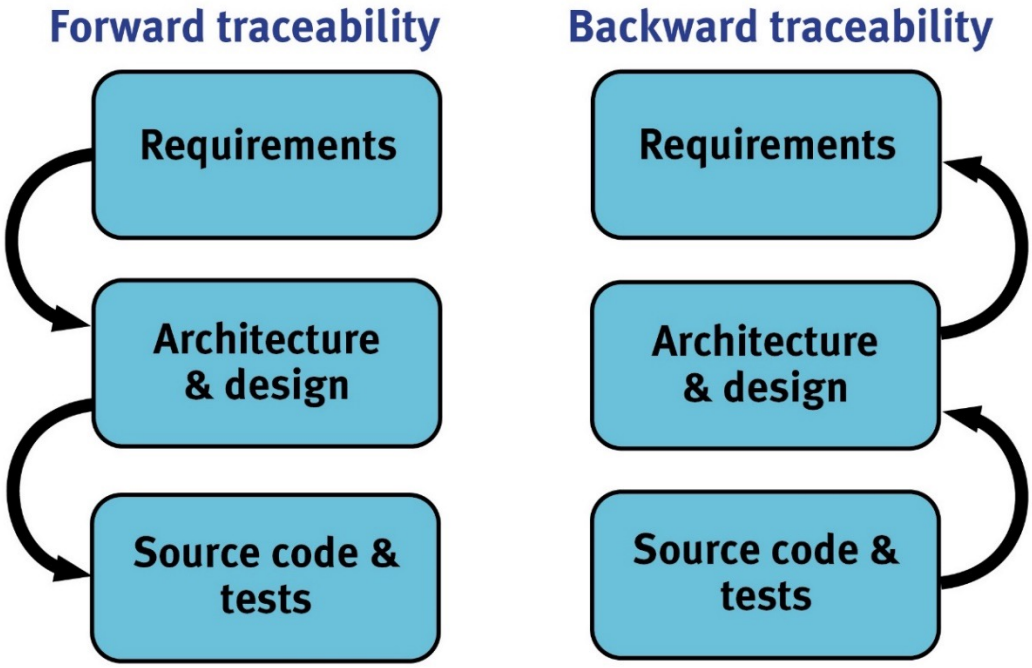


Figure 7: Bi-directional traceability.

Structural coverage analysis (code coverage, data coupling and control coupling) quantifies the extent to which the source code of a system has been exercised by the testing process. Using these practices, it is possible to ensure that code has been implemented to address every system requirement and that the implemented code has been tested to completeness.

The use of software tools offers particularly significant benefits during software development and software verification as discussed in §5.0 and §6.0 of the standard, respectively.

DO-178C Section 5.0: SOFTWARE DEVELOPMENT PROCESSES

Five high-level processes are identified in the DO-178C SOFTWARE DEVELOPMENT PROCESSES section: Software Requirements Process (§5.1), Software Design Process (§5.2), Software Coding Process (§5.3), Integration Process (§5.4), and Software Development Process Traceability (§5.5).

The ideal tools for requirements management (§5.1) depend largely on the scale of the development. If there are few developers in a local office, a simple spreadsheet or Microsoft Word [42] document may suffice. Bigger projects, perhaps with contributors in geographically diverse locations, are likely to benefit from an application life cycle management tool such as IBM Engineering Requirements Management DOORS Family [43], Siemens Polarion ALM [44], or more generally, similar tools offering support for the standard Requirements Interchange Format ReqIF [45].

The products of the design phase (§5.2) potentially include products of Model Based Design (such as system and design models), spreadsheets, textual documents, and many other artifacts. A variety of tools can be involved in their production.

As part of §5.3, DO-178C specifies that certain software coding process objectives must be met. For example, developers of source code should implement low-level requirements and conform to a set of software coding standards.

Further definition of the term “software coding standards” in this context is provided in §11.8 of DO-178C:

- “Programming language(s) to be used and/or defined subset(s). For a programming language, establish an approach to unambiguously define the syntax, the control behaviour, the data behaviour and side-effects of the language. This may require limiting the use of some features of a language.”
- “Source code presentation standards including line length restriction, indentation, and blank line usage.”
- “Source code documentation standards, for example, name of author, revision history, inputs and outputs, and affected global data.”
- “Naming conventions for components, subprograms, variables and constants.”
- “Conditions and constraints imposed on permitted coding conventions, such as the degree of coupling between software components and the complexity of logical or numerical expressions and rationale for their use.”
- “Constraints on the use of coding tools.”

Static analysis

LDRA static analysis tools automate the “inspection” of the source code. They make compliance checking easier, less error prone, and more cost effective by comparing the code under review with the rules dictated by the chosen a software coding standard (Figure 8). Non-conformances are highlighted as required by §6.4.3d of the standard. The tools can also assess the complexity of the code under review to ensure that it stays below a safe threshold for the system, and its data flow analysis facility can be used to identify any uninitialized or unused variables and/or constants as specified by §6.4.3.f.

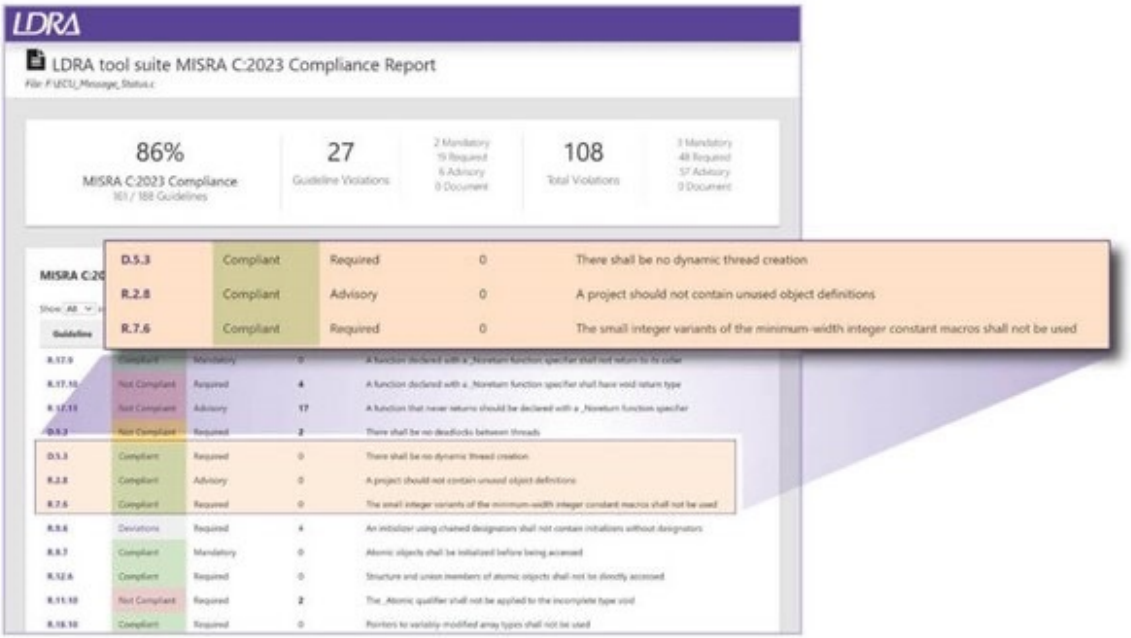


Figure 8: Checking for compliance with the MISRA C:2023 coding standard using the LDRA tool suite.

There are many pre-defined language subsets (sometimes called “coding standards”) available for C, C++ and Ada languages (sidebar). A corporate- or project-specific subset is also acceptable which could be established entirely from scratch or, more pragmatically, based on an established subset with modifications to suit a particular application. It is important that a deployed static analysis tool should be similarly flexible.

While static analysis is valuable for catching many types of errors as code is written, some issues only become apparent during integration. These include interface mismatches, timing issues, and unexpected interactions between modules. DO-178C §5.4 emphasizes the importance of static (and dynamic) analysis to ensure comprehensive verification.

In §5.5, DO-178C mandates that the correctness of the requirements-based development and verification process is determined by requirements coverage or traceability. This analysis assures that requirements are bi-directionally associated between system and high-level requirements, high-level and low-level requirements, and finally low-level requirements and source code.

Coding standards

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

C

MISRA C:2023 [58]
CERT C [56]
CWE [57]

C++

MISRA C++:2023 [59]
CERT C++ [55]
CWE
JSF++ AV [60]
HIC++ [61]

Ada

Ravenscar [62]
Spark 2014 [63]

Java

CWE
CERT J
ESA BSSC

Keeping track of progress

If everything follows the development life cycle in textbook fashion, that is perhaps a one-off, trivial task. The requirements will never change, and tests will never throw up a problem. But unhappily, that is rarely the case.

Consider, then, what happens if a problem is highlighted:

- Perhaps there is a contradiction in the requirements. If so, the requirements will need to change. What other parts of the software are affected?
- Maybe there is a low-level requirement that is not traceable through to a high-level requirement. What needs to change to resolve that?
- Maybe an end user has a new requirement. What impact will that have on established requirements, design and source code?

The TBmanager component of the LDRA tool suite is a desktop traceability application, and is integrated with code review, data and control coupling analysis, low-level testing, and code coverage tools. It eases the project management challenges associated with such complexity by constantly maintaining a requirements traceability matrix down to the source code and test cases - even through disparate repositories (Figure 9). It creates a common user experience across the development team for complete workflow management, creating and collating evidential artefacts which may be used for certification purposes. It links standard objectives, project requirements, design, source code, tests, analyses, and associated artefacts within the software development life cycle.

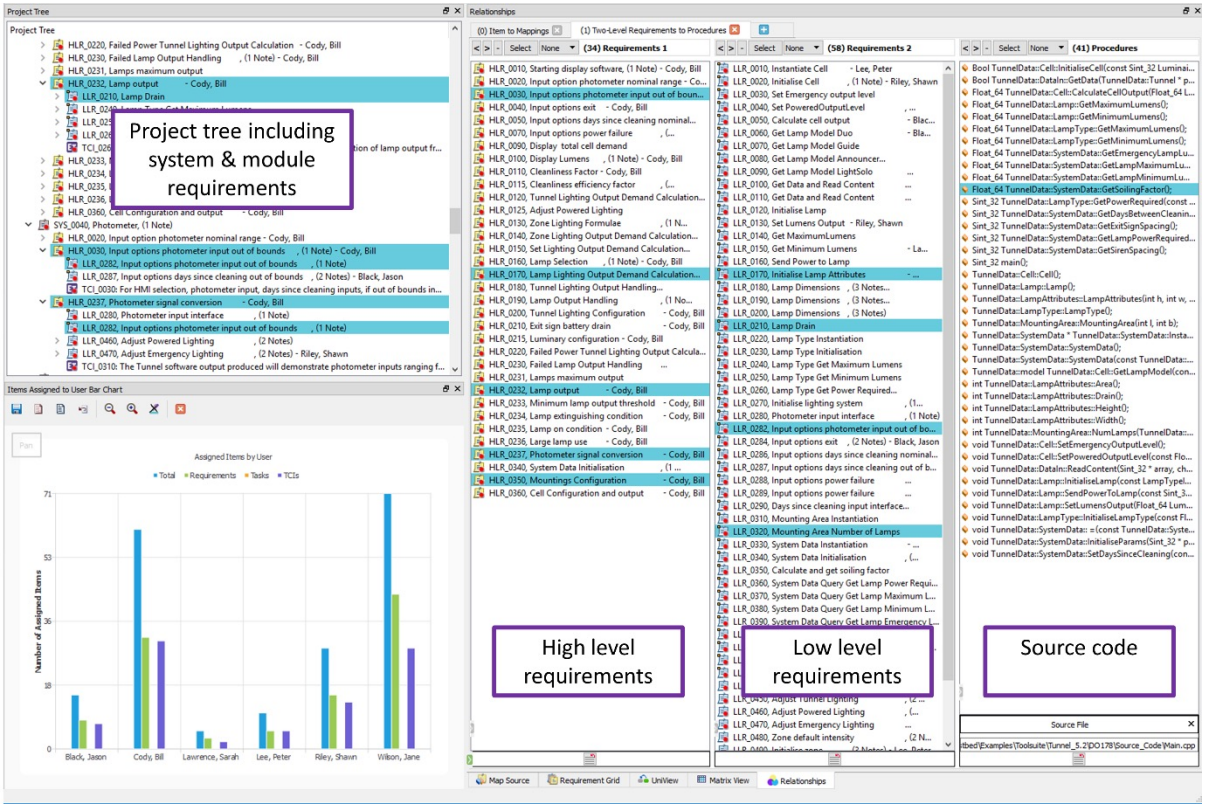


Figure 9: Automating requirements traceability with the TBmanager component of the LDRA tool suite.

LDRAvault is complementary to TBmanager. It is a web-based, enterprise level application that aggregates and manages certification artifacts across projects and programs providing transparency into the development and verification process. Data access can be easily controlled, and analysis and compliance results can be easily shared with pertinent parties including regulatory authorities. Examples include code reviews, code coverage analysis, and unit testing results.

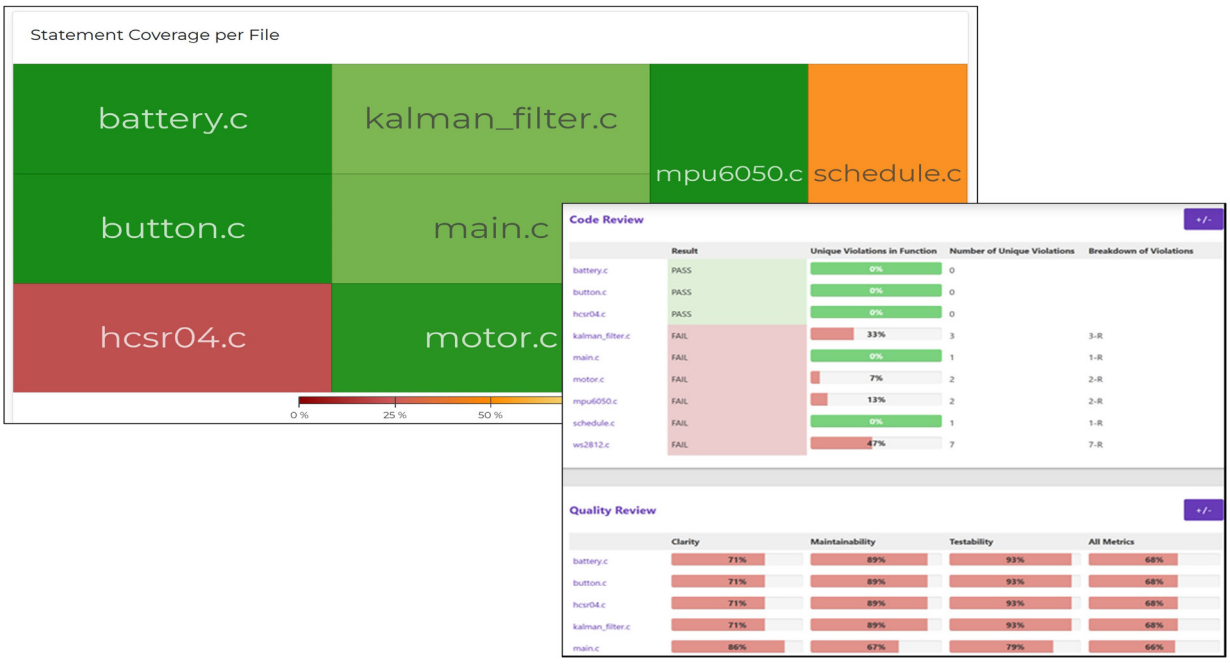


Figure 10: Aggregating certification evidence with LDRAvault.

Furthermore, imported reports and results across multiple users and projects are collected into snapshots. Using the snapshots, LDRAvault generates visualizations such as heat maps and trend graphs that users can explore to discover the root causes of issues.

DO-178C Section 6.0: SOFTWARE VERIFICATION PROCESSES

DO-178C §6.0: SOFTWARE VERIFICATION PROCESSES outlines the activities and criteria required to ensure that the software performs its intended functions correctly and safely, and that it meets the established requirements and design standards.

Dynamic analysis

In contrast to static analysis which can be thought of as an automated “inspection” of the source code, dynamic analysis involves executing the Executable Object Code (EOC) piecemeal or in its entirety. The aim is to demonstrate the correct functionality of the entire source code base in its native environment, and so DO-178C requires the use of a target environment representative of that to be deployed in the completed application.

Structural coverage analysis

Dynamic analysis is used to demonstrate correct functionality, and to show that all parts of the code base have been exercised - that is, to demonstrate adequate “structural coverage”. Structural coverage analysis requires both static analysis to expose the underlying structure of the software, and dynamic analysis to reveal which parts of it have been exercised and which have not.

Low-level, integration, and system testing

DO-178C identifies objectives to achieve test coverage of high- and low-level requirements, and to achieve appropriate test coverage of the software structure and of the data and control coupling. The “test cases and procedures” referenced in the standard as means to achieve these aims may include low-level tests (sometimes referred to as unit tests), integration tests, or system tests, and probably a combination of all three.

Low-level tests are designed to verify the implementation of low-level requirements. Test procedures need to be authored, reviewed, and executed to ensure the software does not contain any undesired functionality. Low-level tests can then be executed on the target hardware or simulated environment as specified in the Software Verification Plan (SVP). Once the test procedures are executed actual outputs are captured and compared with the expected results, and pass/fail results reported.

Software integration testing is designed to verify the interrelationships between the software components with respect to both the requirements and the software architecture. In practice, the mechanisms used for low-level testing are often extended to verify behaviour in a call tree.

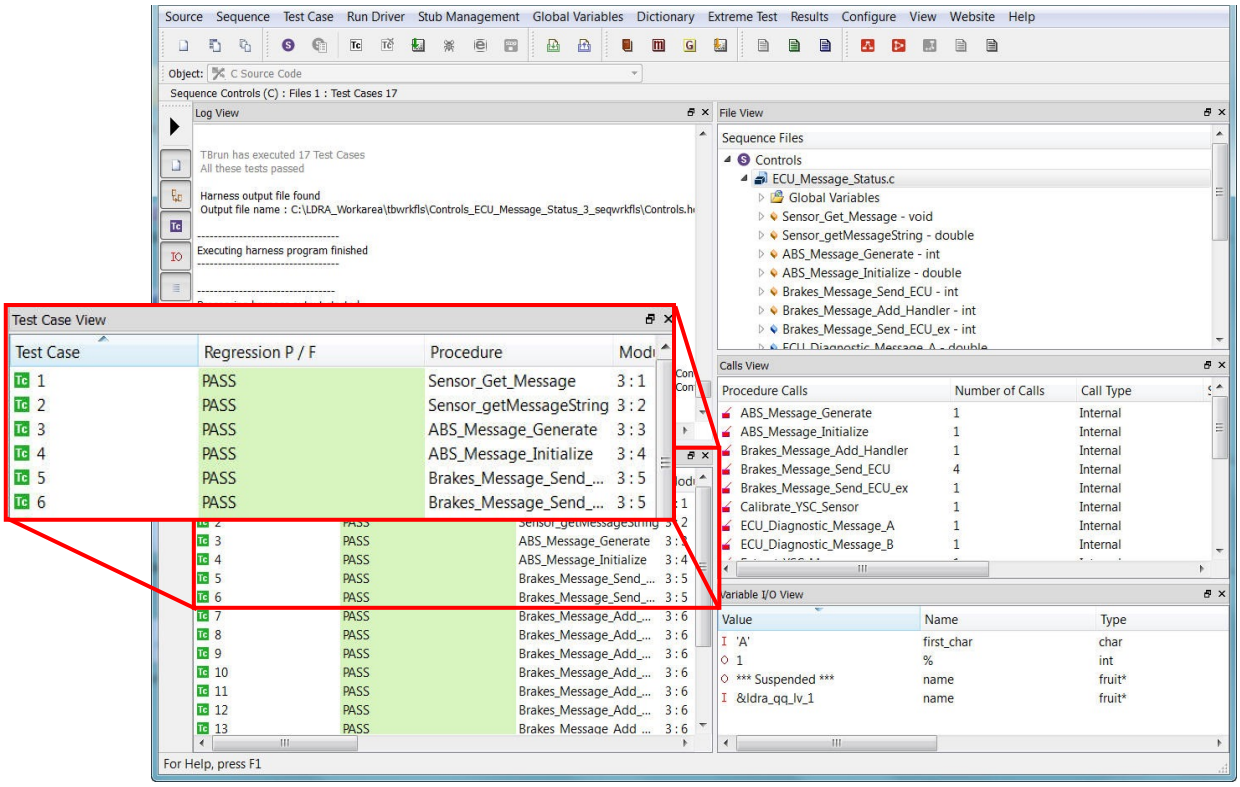


Figure 11: Automating low-level and integration testing with the TBrun component of the LDRA tool suite [46].

Should changes become necessary – perhaps because of a failed dynamic test, or in response to a requirement change - then all impacted low-level and integration tests would need to be re-run. 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.

Keeping track of progress

Just as was the case for development phase, keeping track of a project in flux during the verification phase is challenging. The TBmanager component of the LDRA tool suite automates the maintenance of the bi-directional relationship between the products of the different development phases while saving a great deal of time and helping eliminate errors – not just as far the development of the requirements and source code, but through to requirements-based testing and test coverage for both high- and low-level requirements.

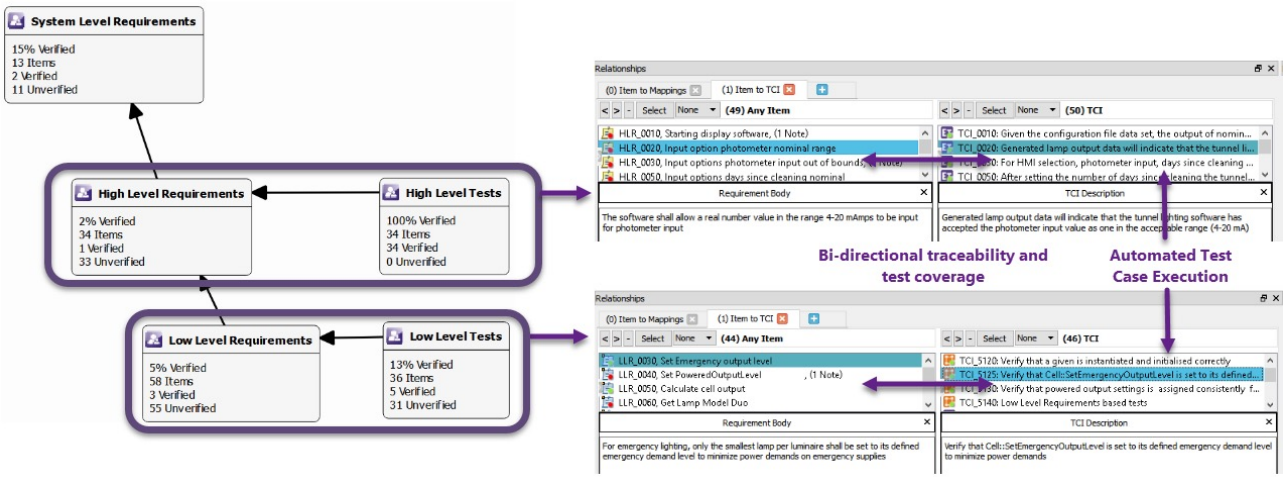


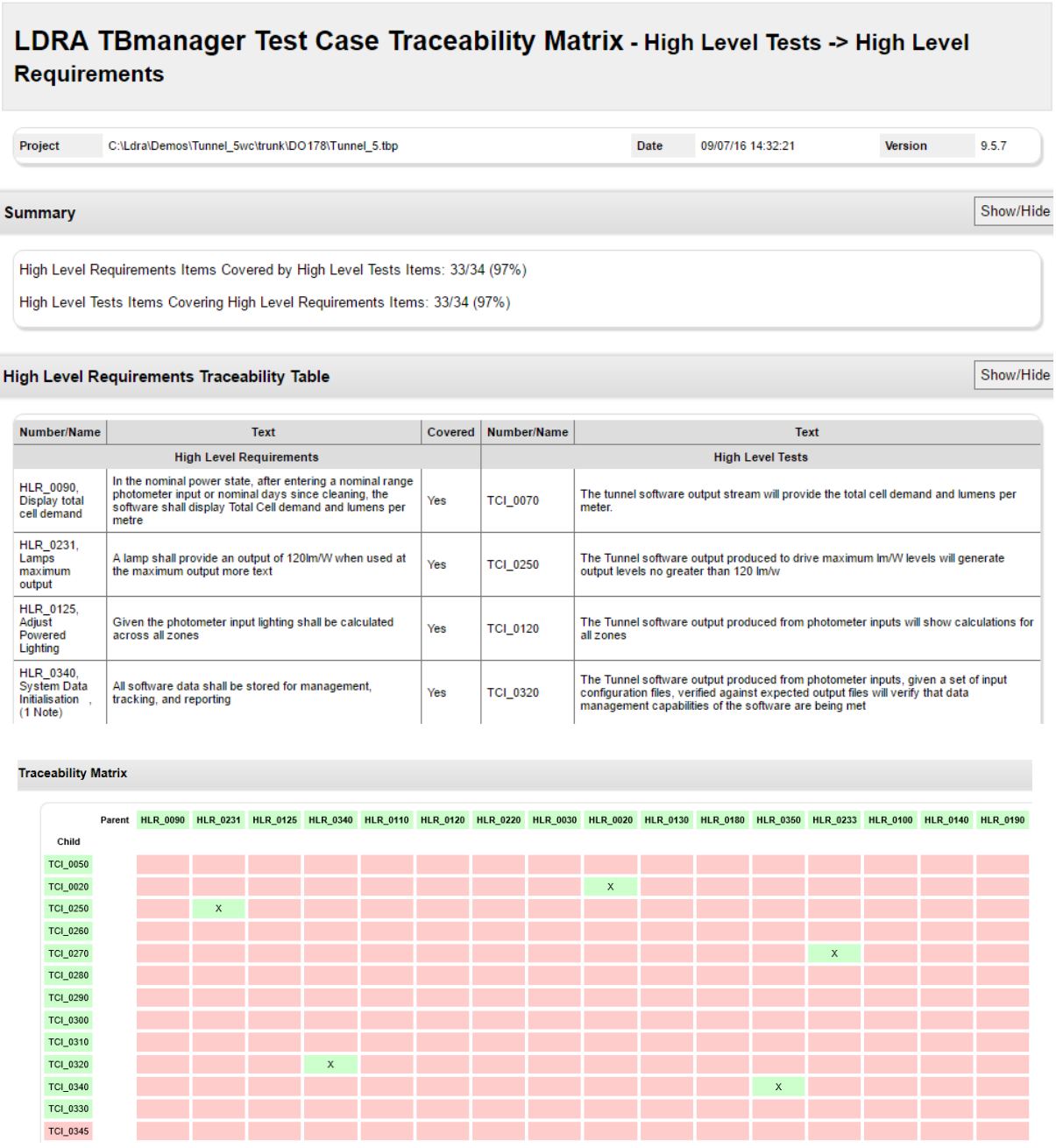
Figure 12: Tracing requirements with the Graphical User Interface (GUI) of TBmanager, a component of the LDRA tool suite

The area on the left of (Figure 12) shows how a graphical representation of the traceability policy between requirements and tests cases of different scopes. Requirements and test cases are then authored or imported and linked together, so completeness in requirements decomposition and test coverage can be assessed. Test cases can be reviewed and developed based on requirements identifying and filling gaps in requirements coverage quickly and easily.

Bi-directional analysis can also identify uncovered "orphan" test cases that are not linked to requirements, highlighting the need to make appropriate changes to requirements, test cases, or traceability relationships. It can provide impact analysis in relation to changes in requirements, tests, or code and their potential effects on timescales. It can provide the intelligence for regression testing to be targeted, minimizing the incremental verification and review burden. And it can provide required evidential artifacts such as traceability matrices, to show that test coverage of high-level and low-level requirements has been achieved[2].

(Figure 13) shows a traceability matrix between high-level requirements and functional test cases. In this example, thirty-three out of thirty-four requirements have an associated test case, so the test coverage of high-level requirements objective is still unfulfilled. This level of transparency is essential to ensure that all requirements have associated test cases.

[2] Table A-7 objectives 3 and 4 from RTCA DO-178C, Copyright © 2011 RTCA, Inc. All rights acknowledged.



LDRA TBmanager Test Case Traceability Matrix - High Level Tests -> High Level Requirements

Project	C:\Ldra\Demos\Tunnel_5wc\trunk\DO178\Tunnel_5.tbp	Date	09/07/16 14:32:21	Version	9.5.7

Summary Show/Hide

High Level Requirements Items Covered by High Level Tests Items: 33/34 (97%)

High Level Tests Items Covering High Level Requirements Items: 33/34 (97%)

High Level Requirements Traceability Table Show/Hide

Number/Name	Text	Covered	Number/Name	Text
High Level Requirements			High Level Tests	
HLR_0090, Display total cell demand	In the nominal power state, after entering a nominal range photometer input or nominal days since cleaning, the software shall display Total Cell demand and lumens per metre	Yes	TCI_0070	The tunnel software output stream will provide the total cell demand and lumens per meter.
HLR_0231, Lamps maximum output	A lamp shall provide an output of 120lm/W when used at the maximum output more text	Yes	TCI_0250	The Tunnel software output produced to drive maximum lm/W levels will generate output levels no greater than 120 lm/w
HLR_0125, Adjust Powered Lighting	Given the photometer input lighting shall be calculated across all zones	Yes	TCI_0120	The Tunnel software output produced from photometer inputs will show calculations for all zones
HLR_0340, System Data Initialisation , (1 Note)	All software data shall be stored for management, tracking, and reporting	Yes	TCI_0320	The Tunnel software output produced from photometer inputs, given a set of input configuration files, verified against expected output files will verify that data management capabilities of the software are being met

Traceability Matrix

Child \ Parent	HLR_0090	HLR_0231	HLR_0125	HLR_0340	HLR_0110	HLR_0120	HLR_0220	HLR_0030	HLR_0020	HLR_0130	HLR_0180	HLR_0350	HLR_0233	HLR_0100	HLR_0140	HLR_0190
TCI_0050																
TCI_0020									X							
TCI_0250		X														
TCI_0260																
TCI_0270													X			
TCI_0280																
TCI_0290																
TCI_0300																
TCI_0310																
TCI_0320				X												
TCI_0340												X				
TCI_0330																
TCI_0345																

Figure 13: Excerpt from a traceability matrix (High-level requirements to tests cases) as illustrated by TBmanager, a component of the LDRA tool suite

At the enterprise level, LDRAvault is as applicable to verification as it is to development. Visualizations like heat maps and trend graphs are leveraged to illustrate the progress of the verification tasks associated with the project.

Structural coverage analysis objectives

Structural Coverage (SC) identifies which code structures and component interfaces have been exercised during the execution of requirements-based test procedures, facilitating the empirical measurement of requirements-based test effectiveness. As the name implies, Structural Coverage Analysis (SCA) involves the scrutiny of the SC to determine if there are any parts of the code which have not been sufficiently exercised, and if not, why.

DO-178C §6.4.4 details requirements for the achievement of 100% MC/DC, decision, and statement coverage, depending on DAL. Collation of structural coverage metrics is typically achieved by "instrumenting" a copy of the source code (that is, superimposing function calls to collate coverage

data) and executing that instrumented code using requirement-based test cases. These test cases primarily reference high-level requirements, supplemented by low-level requirements as needed.

SCA is then applied to assess the effectiveness of this testing by measuring how much of the code has been exercised. Coverage of portions of code unexecuted thus far may require additional test cases or modifications to existing test cases, changes to requirements, removal of dead code, or perhaps the identification of deactivated code and resulting unintended functionality. An iterative "review, analyse, verify" cycle is typically needed to ensure that the software coverage objectives are fulfilled, and graphical representation can be a great help.

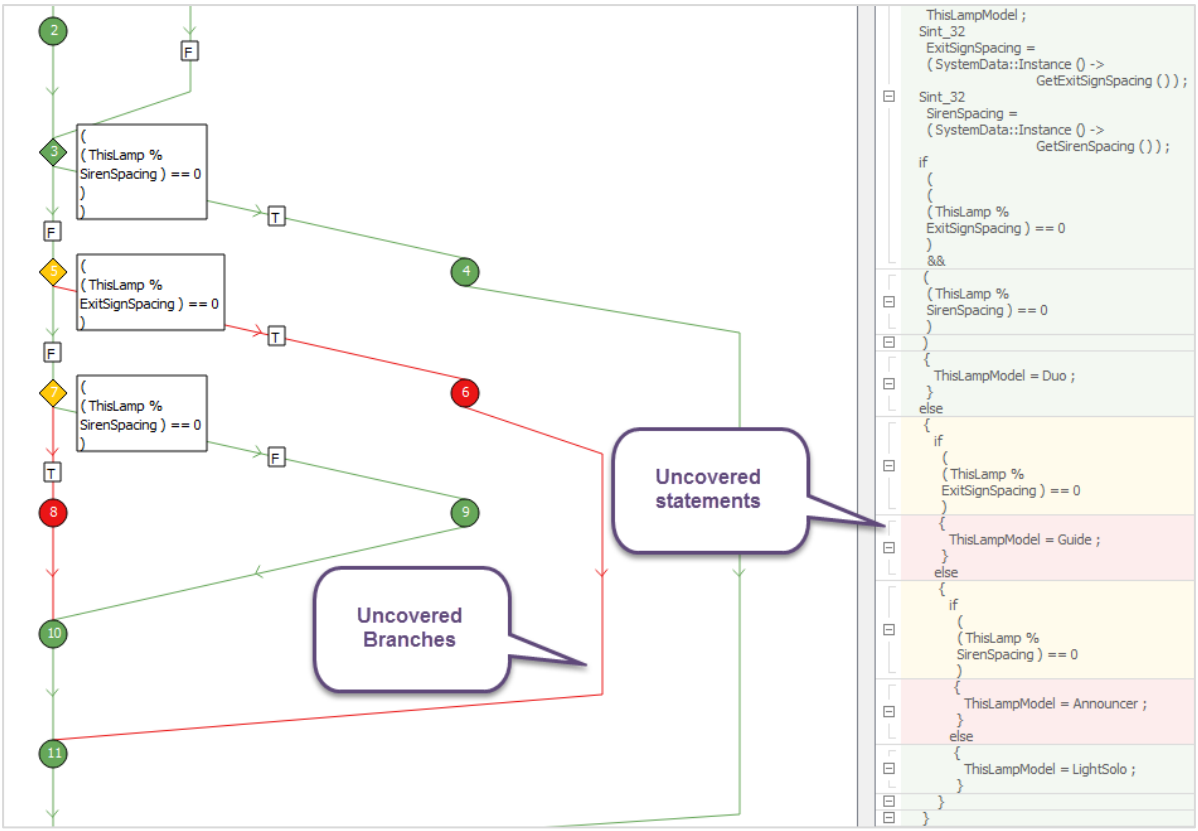


Figure 14: Graphical visualisation of code coverage in a flow graph in the LDRA tool suite.

System requirements can be shown to have been correctly decomposed, implemented, and verified by combining a complete trace from requirements through to code and test cases, with the achievement of comprehensive functional test coverage and structural coverage objectives.

Modified Condition / Decision Coverage (MC/DC)

In addition to statement and branch coverage, for level A software MC/DC is obligatory. MC/DC requires that "Each condition in a decision has been shown to independently affect that decision's outcome" (DO-178C Glossary).

It might be supposed that where there are multiple conditions in decisions such as that shown in (Figure 15), the right course of action would be to test every possible combination. However, as soon as there are more than two or three conditions involved, the amount of testing that would require becomes prohibitive.

```
void check_change_gear (int *cg,int tg,int *rpm,int *cp,int mc,int fp)
{
   if (((*rpm >= M1) && (tg > *cg)) || ((*rpm <= M2) && (tg < *cg)) ||
        (mc == 1) || (fp < M5))
   {
```

Figure 15: An example of a multiple condition decision

MC/DC is a coverage metric for multiple condition decisions. It does not require every possible combination to be executed but instead requires only one more test than the number of conditions involved. In the example shown there are 6 conditions, and so a total of 64 possible combinations – and yet only 7 tests are needed to achieve MC/DC coverage.

However, those tests must show that each of the conditions independently affect the result, and it is not always obvious which input values might achieve that. The use of an MC/DC planner makes the selection of appropriate values much easier.

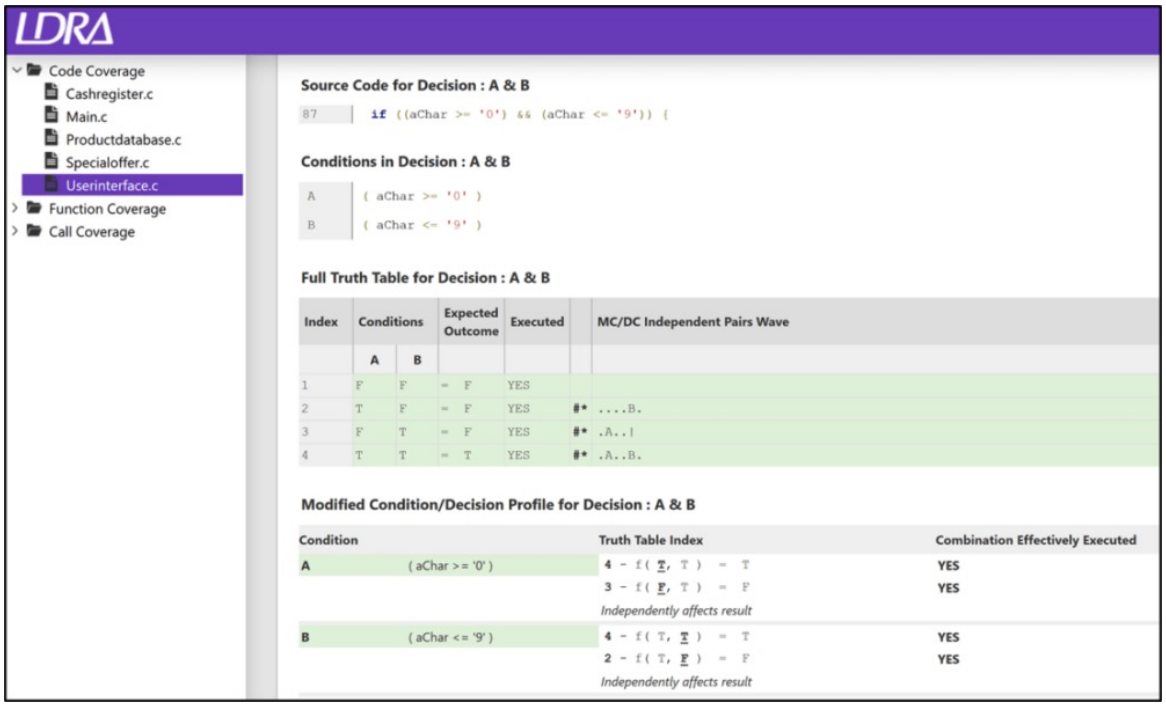


Figure 16: Using an MC/DC planner "decision truth table" in the LDRA tool suite

Source to object code traceability

For Level A systems, structural coverage at the source level isn't enough. Compilers often add additional code or alter control flow, and often their behaviour is not deterministic. To ensure that functionality is not compromised, DO-178C §6.4.4.2.b states that:

"...if the software level is A and a compiler, linker, or other means generates additional code that is not directly traceable to Source Code statements, then additional verification should be performed to establish the correctness of such generated code sequences."

An automated mechanism to provide evidence of that verification can make this process much more efficient. Because there is a direct one-to-one relationship between object code and assembly code, one way for a tool to represent this is to display a graphical representation of the source code alongside the equivalent representation of the assembly code. Object Code Verification (OCV) measures code coverage at both the source and the assembly level by instrumenting each in turn (Figure 17).

This approach provides a means for the demonstrable and deterministic verification of the Executable Object Code (EOC) on the target system. For OCV to be effective, it therefore needs to support the microprocessor, associated instruction set, and compiler deployed on that system.

Three discrete modes are used for each test case to quickly identify the “additional code” referenced in the standard and dramatically reduce laborious manual analysis.

1. The test case is executed without instrumentation to confirm correct functionality.
2. The test case is executed by leveraging instrumentation at the source code level.
3. Finally, the test case is executed with instrumentation at the assembly code level to identify any uncovered statements or branches that may have been inserted or altered during the compilation and linking process.

Typically, a few more requirements-based tests can be added to verify this additional code to meet the objectives of §6.4.4.2.b. .

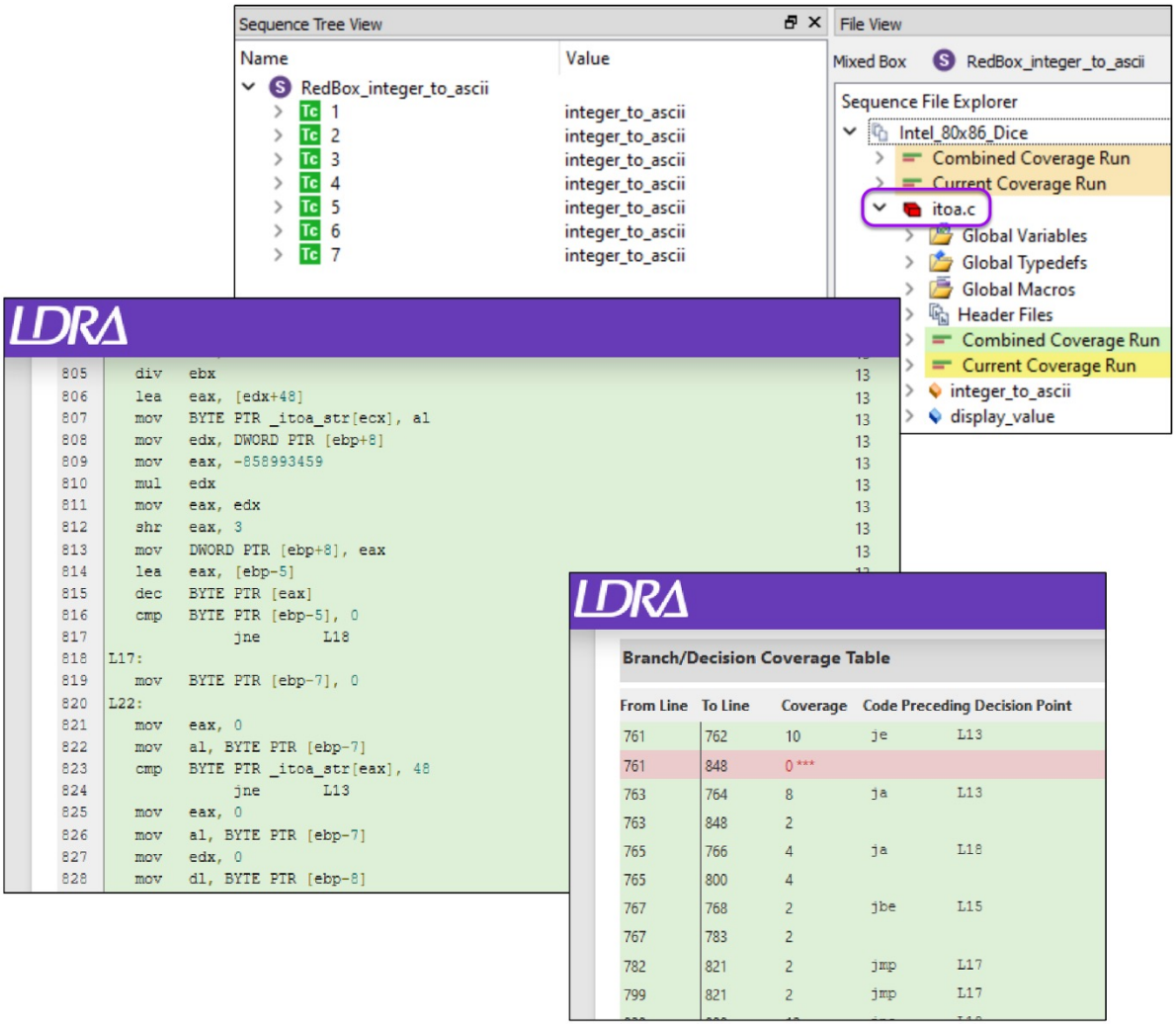


Figure 17: Identifying incomplete object code coverage using the LDRA tool suite.

Automated source code instrumentation and coverage data analysis reduces space and time overhead which in turn makes the technology scalable and adaptable to a wide array of cross compilers, targets, in-circuit emulators, and other embedded environments. Target integrations are highly extensible and support processors from simple 8-bit devices to high-performance multicore architectures, IDEs, and I/O integrations.

Data Coupling and Control Coupling

DO-178C §6.4.4.2.c requires "analysis to confirm that the requirements-based testing has exercised the data and control coupling between code components."

Data coupling and control coupling analysis therefore need to be performed post execution, and the generated artifacts reviewed against the system requirements and architecture.

The scope of coupling analysis can range from coupling within a file, through file-to-file coupling, to module-to-module coupling. As the name implies, file-to-file coupling refers to the dependencies between different files in a software system (such as functional calls from one file to another, or shared data). Module-to-module coupling broadens the concept to larger units of code that can consist of multiple files.

The analysis of control and data coupling by traditional means can be tedious and challenging. LDRA's patented approach leverages static analysis and dynamic analysis in tandem to make that process more efficient and less error prone.

Control Coupling

Control Coupling is defined in the glossary of DO-178C as "The manner or degree by which one software component influences the execution of another software component."

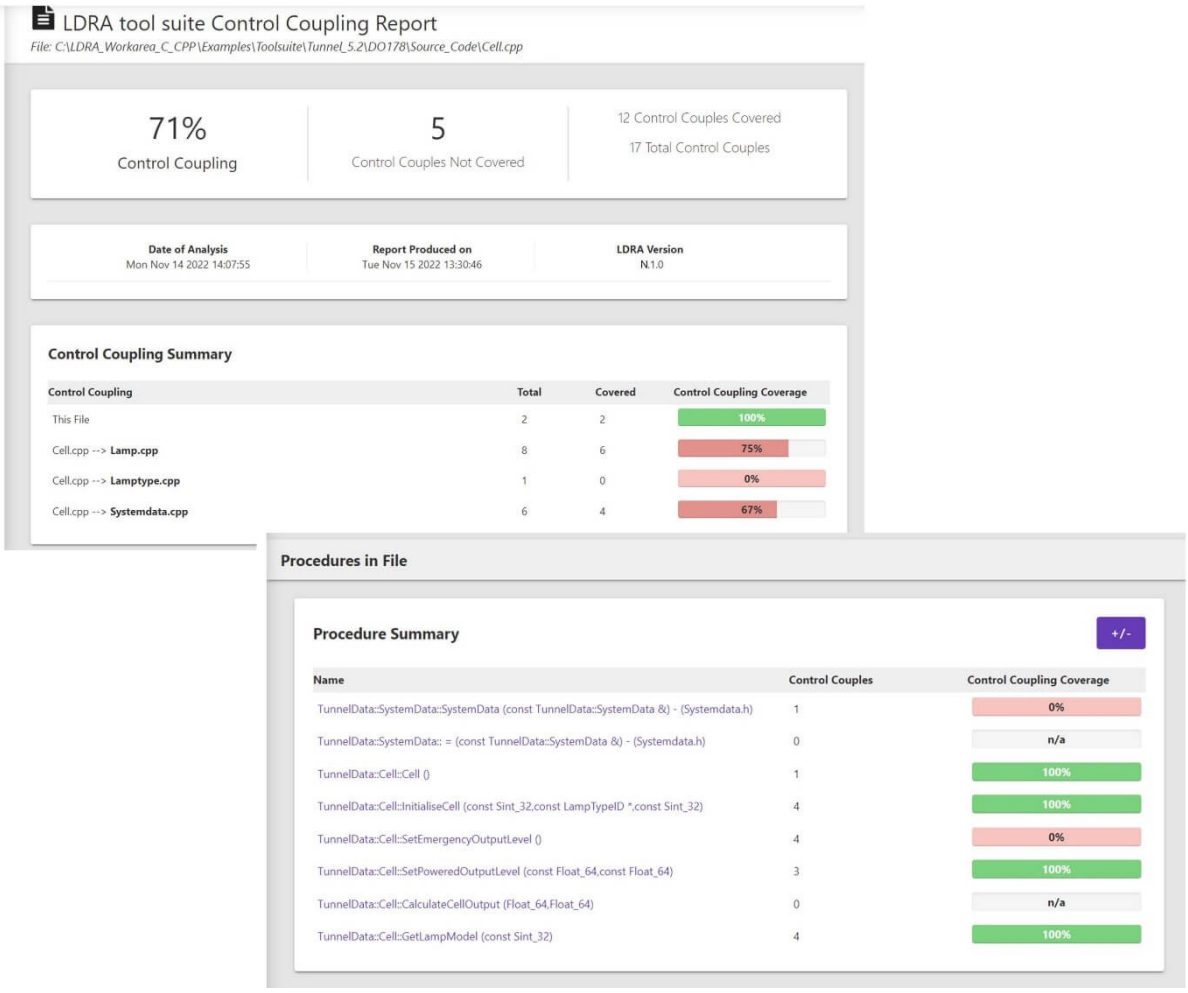


Figure 18: Procedure/function call coverage as seen in call graphs generated by the LDRA tool suite.

The uppermost image in (Figure 18) shows a report where 17 control calls have been identified, with 2 relating to intra-file calls (within this file) and the remainder relating to external calls. The lower image shows a report that identifies the procedures within a particular file, the control couples associated with

procedures, and the extent to which they have been exercised.

These reports help to identify any gaps and guide targeted verification activities, and ultimately provide evidence that all couples have been validated.

Data Coupling

Data coupling is defined in the glossary of DO-178C to be "The dependence of a software component on data not exclusively under the control of that software component". DO-178C §6.4.4.2.c requires "Analysis to confirm that the requirements-based testing has exercised the data and control coupling between components".

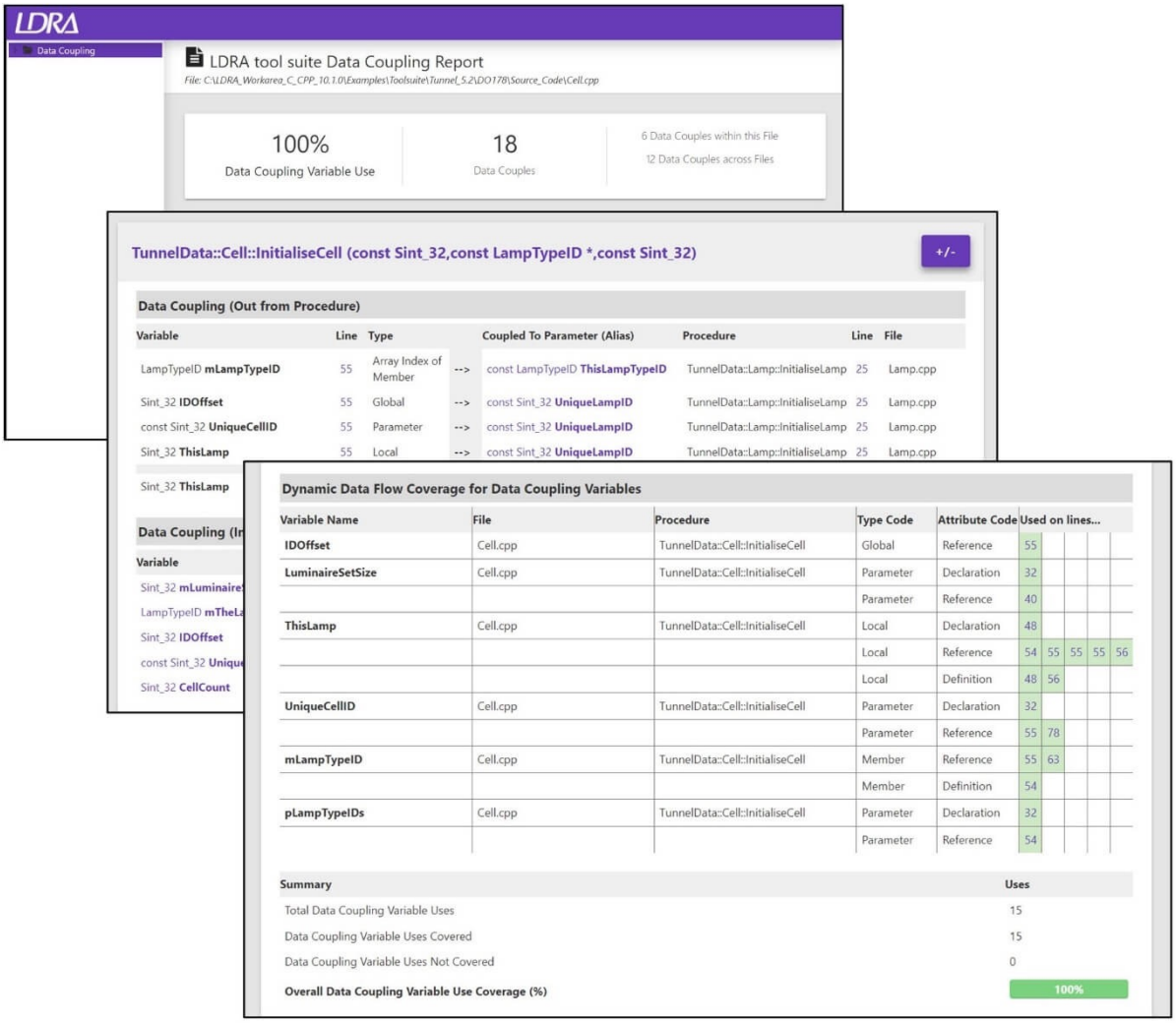


Figure 19: Data coupling analysis with the LDRA tool suite

Data coupling analysis is focused on the observation and analysis of data elements as they are set and used ("set/use pairs") across software component boundaries. Manually performing these exercises with debuggers is labour intensive, difficult to repeat, and error prone. Automating the activity dramatically reduces that overhead.

Execution time

DO-178C requires that software execution times be predictable and consistent with the system's requirements, referencing a need for execution time analysis throughout §6.4. Examples include:

- **§6.4.4.3:** "Test results should confirm that the software meets its execution timing requirements."
- **§6.4.4.3.b:** "Structural coverage analysis should confirm that the worst-case execution time is met and does not exceed system limits."
- **§6.4.4.3.c:** "Tests should be performed to ensure that the software does not exceed the specified execution time limits and operates within the defined timing constraints."

The aim for the developer is to establish an upper bound for each task (called Worst Case Execution Time, or WCET) that the actual execution time should never exceed. The WCET will depend on both the target hardware and the software.

A commonplace traditional approach to determining the Worst Case Execution Time (WCET) for a single-core processor involves estimating the execution time by examining the source code and calculating the longest possible execution path. This incorporates path analysis to identify the longest path, cache analysis to predict cache behaviour, and loop analysis to estimate the maximum iterations of loops. The goal is to provide a conservative yet accurate WCET estimate so that resources can be allocated to ensure that the actual execution time will not exceed this prediction.

However, it is increasingly common practice to deploy multicore processors (MCPs) in avionics applications. These MCP devices almost always share hardware resources outside the processor cores, and time-related delays occur as users wait for access to these Hardware Shared Resources (HSRs). (Figure 20) shows the example of shared cache, but most devices have very many HSRs to contend with.

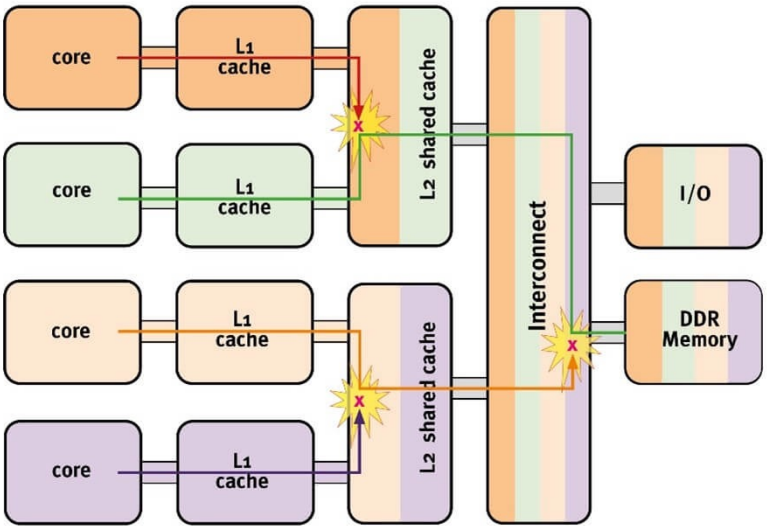


Figure 20:The sharing of resources leads to unpredictable delays as users wait for access.

CAST-32A, AC 20-193, and AMC 20-193

CAST-32A [47] was an informational position paper written "to identify topics that could impact the safety, performance and integrity of a software airborne system executing on Multi-Core Processors (MCP)". It was developed to offer guidance in addition to that provided by existing standards such as DO-178C which includes nothing specific to the use of MCPs.

The principles detailed in CAST-32A have since been formally adopted into the FAA's AC 20-193 document and EASA's AMC 20-193 equivalent. They are known collectively as A(M)C 20-193.

The LDRA tool suite facilitates a practical, CAST-32A & A(M)C 20-193 compliant approach to addressing the multicore WCET problem. It involves the optimization of system configuration by means of interference research through measured execution times using the TBrun component of the LDRA tool suite supported by the optional TBwcet module [48].

Three of the documents' clauses are particularly significant:

MCP_Resource_Usage_3

"The applicant has identified the interference channels that could permit interference to affect the software applications hosted on the MCP cores, and has verified the applicant's chosen means of mitigation of the interference"

This places the onus on the developer to demonstrate that all interference channels are known and have been adequately dealt with.

MCP_Resource_Usage_4

"The applicant has identified the available resources of the MCP and of its interconnect in the intended final configuration, has allocated the resources of the MCP to the software applications hosted on the MCP, and has verified that the demands for the resources of the MCP and of the interconnect do not exceed the available resources when all the hosted software is executing on the target processor."

The need for the analysis of execution times is implicit in this objective.

MCP_Software_1

"Applicants who have verified that their MCP Platform provides both Robust Resource and Time Partitioning … may verify applications separately on the MCP and determine their WCETs separately."

As the documents highlight there are many highly effective robust partitioning mechanisms available. These mitigate for many of the most significant interference channels. However, there are two caveats.

- The robust partitioning offered by an RTOS, Separation Kernel, or Hypervisor is only as effective as its configuration.
- Perhaps more significantly, not even excellent robust partitioning schemes can cleanly partition all Hardware Shared Resources (HSRs) because for most MCPs, many HSRs are not allocated until they are first accessed.

Whether robust partitioning is in place or not, the onus remains on the developer to demonstrate that interference mitigation is effective which can only be achieved through measurement.

LDRA's solution adopts a "wrapper" principle which affords the flexibility to perform execution time analysis from complete system behaviour, through a thread or process, right down to class/function/procedure level. This approach provides the ability to both "drill in" to problem areas, and to analyse the complete system (Figure 21).

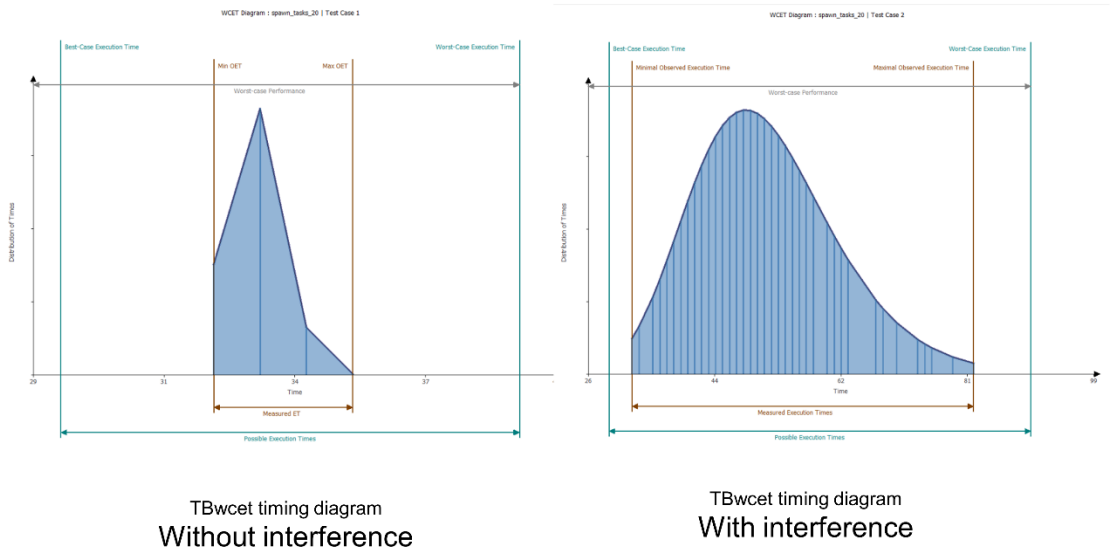


Figure 21: Interference research and timing analysis with the LDRA tool suite.

Stressors consisting of specially designed software are essential in MCP applications for use in the derivation of WCET by simulating worst-case interference scenarios. They allow the developer to evaluate the impact of resource contention and hence to ensure that the timing analysis models accurately reflect the maximum possible execution times under various stress conditions.

LDRA leaves the choice of HSR stressor mechanism to the user. Open source solutions such as Stress ng [49] provide custom stressors for HSRs. The flexibility of the test harnesses in the LDRA tool suite presents the opportunity for these stressors to be adapted to suit specific circumstances, or for new ones to be developed from first principles.

The resulting empirical nature of verification and validation in the multiprocessor environment as compared to single core is illustrated in makes it imperative that project managers can adapt readily to changing requirements and configurations. (Figure 22) shows a proposed iterative interference research process. [50]

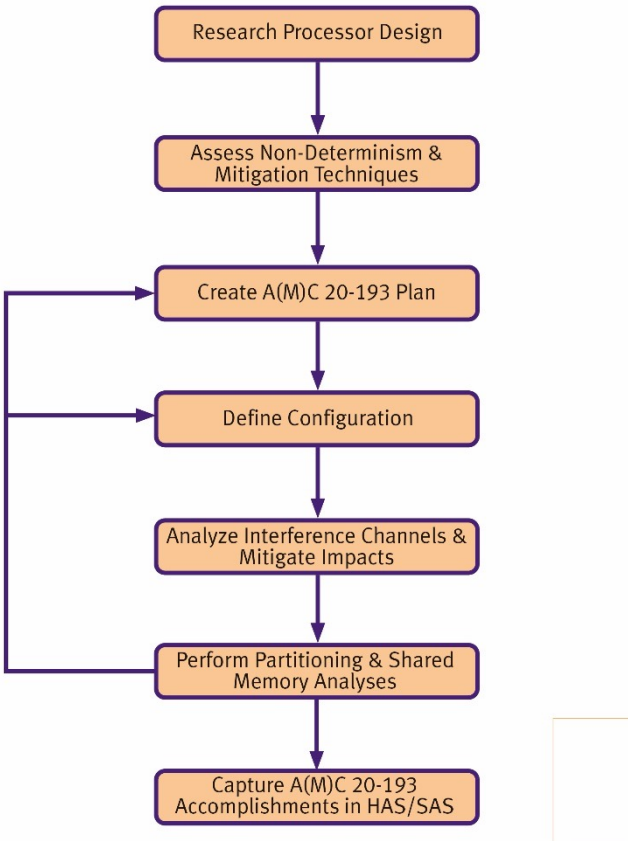


Figure 22: Interference research for MCPs is iterative by nature

It is highly likely that such interference research will lead to changes in system or software requirements, and conversely that changes in system functional requirements will drive new interference channels or affect existing ones. Under these circumstances, an automated mechanism is vital to keep track of what needs revisiting for renewed verification and validation, and hence to keep the project on schedule, and within budget. The TBmanager component of the LDRA tool suite provides just such a mechanism.

Supplements to DO-178C

When DO-178C superseded DO-178B, four supplements were created to address the need for clarity and guidance on modern software development practices within the context of aviation software certification. They add, delete, or modify objectives, activities, and life cycle data in DO-178C to cater to the specific needs of these technologies.

DO-330 is not quite like the others. It discusses software tool qualification primarily in the context of avionics related software, but equally it is design to be applicable to other safety-critical sectors. DO-331, DO-332, and DO-333 are each applied when the corresponding development technique is used, enhancing the DO-178C framework to cover a broader range of software development methodologies.

The EUROCAE equivalents to the four documents are ED-215, ED-216, ED-217, and ED 218, respectively (Figure 23).

RTCA identifier	EUROCAE identifier	Title
DO-330	ED-215	Software Tool Qualification Considerations
DO-331	ED-216	Model-Based Development and Verification Supplement to DO 178 and DO-278A
DO-332	ED-217	Object-Oriented Technology and Related Techniques Supplement to DO-178 and DO-278A
DO-333	ED-218	Formal Methods Supplement to DO-178 and DO-278A

Figure 23: Supplement identifiers

DO-330 Software Tool Qualification Considerations

If software tools are to automate significant numbers of DO-178C activities while producing evidential artifacts showing that objectives have been met, it is essential to ensure that those tools can be relied upon within the specific operational context in which the software product will be used. DO-178C states that:

"...the purpose of the tool qualification process is to ensure that the tool provides confidence at least equivalent to that of the processes of this document are eliminated, reduced, or automated."

Tool qualification is a vital part of to the certification process, and it is documented in the supplement Software Tool Qualification Considerations (DO-330)[3].

"DO-330 Software Tool Qualification Considerations" introduces the concept of Tool Qualification level (TQL) based on three criteria:

1) A tool whose output is part of the airborne software and thus could insert an error
2) A tool that automates verification processes and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:
   a) Verification processes other than that automated by the tool, or
   b) Development processes that could have an impact on the airborne software.
3) A tool that, within the scope of its intended use, could fail to detect an error.

Where a tool like the LDRA tool suite is designed to be used for verification purposes, its output is not used as part of the airborne software, and it therefore cannot introduce errors into the software. That makes it a criteria 3 tool. Irrespective of the application DAL, such a tool is always assigned Tool Qualification Level 5 (Figure 24).

[3] RTCA DO-330 Software Tool Qualification Considerations Supplement to DO-178C and DO-278A.

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

Figure 24: Tool Qualification Level Matrix

Certification authorities undertake tool qualification on a project by project basis, so the responsibility for showing the suitability of any tools falls on to the organisation developing the application. However, they can make use of Tool Qualification Support Packages (TQSP) provided by the vendor. Such packages typically contain a series of documents, starting with the Tool Operation Requirements that identify the development process needs satisfied by the tool and including test cases to demonstrate that the tool is operating to specification in the verification environment.

Tool Qualification documentation must be referenced in other planning documents, and it plays a key role in the compliance process (Figure 25).

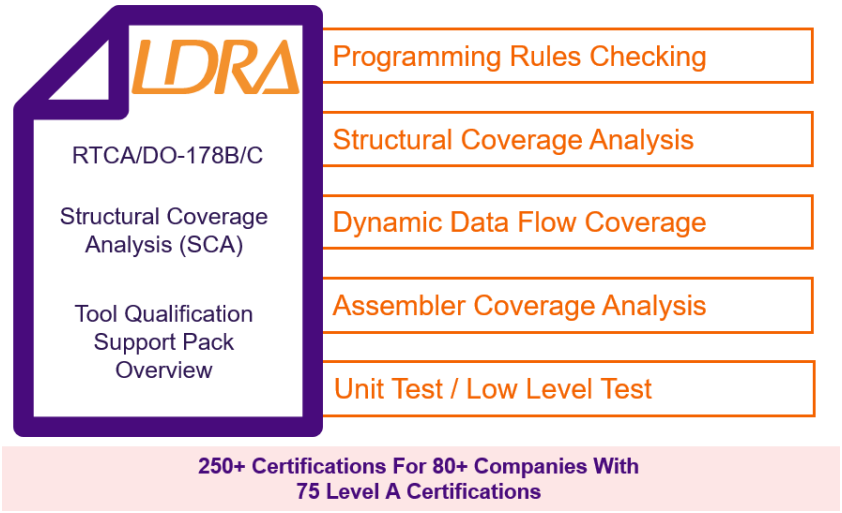


Figure 25: LDRA Tool Qualification Support Packages

DO-331 Model-Based Development and Verification Supplement

Both Model-Based Systems Engineering (MBSE) and Model-Based Development (MBD) are addressed within a supplement called "DO-331 - Model-Based Development and Verification Supplement to DO-178C and DO-278A". MBSE guidance emphasizes the overall system's requirements and interactions, while MBD recommendations concentrate on the software's design and testing processes.

DO-331 takes the approach that specification models or design models take the place of high-level and low-level requirements respectively. Table MB.1-1 (Figure 26) shows that textual requirements may be linked to models upstream or downstream. The standard details some caveats (Notes 1-3) that apply to this table; these are omitted here for brevity.

Process that generates the life cycle data	MB Example 1	MB Example 2	MB Example 3	MB Example 4	MB Example 5
System Requirement and System Design Processes	Requirements allocated to software	Requirements from which the Model is developed	Requirements from which the Model is developed	Requirements from which the Model is developed	Requirements from which the Model is developed
Software Requirement and Software Design Processes	Requirements from which the Model is developed	Specification Model	Specification Model	Design Model	Design Model
	Design Model	Design Model	Textual description		
Software Coding Process	Source Code	Source Code	Source Code	Source Code	Source Code

Figure 26: Model Usage Examples[4]

Popular tools such as MathWorks® Simulink® [51], IBM® Engineering Systems Design Rhapsody® [52], and ANSYS® SCADE suite [53] can generate code automatically. DO 331 §MB.5.0 (Software Development Processes) addresses traceability, model standards and more for both software requirements and design processes where such tools are used. §MB.5.3 (Software Coding Process) is merely a cross-reference the equivalent section in DO 178C, underlining the fact that best-practice coding-related process activities still apply whether code is hand-coded based on a set of textual requirements, hand-coded based on design models, or auto-generated from a tool.

Projects using auto-generated code almost always contain some hand code (that is, handwritten code) too, and often include legacy hand-coded components. It is possible to apply different coding standards to these different code subsets, such as MISRA C:2023 for hand code, MISRA AC:2023 [54] for auto-generated code, and a custom coding standard for legacy code.

DO-331 §MB.6.0 (Software verification process) expands on how best practice applies to MBD, with DO-331 §MB.6.8.2 (Model Simulation for Verification of Executable Object Code) expanding upon which verification objectives can be partially satisfied at the model level, and which must be performed at the target level.

Partial credit in the model

DO-331 §MB.6.8.2 states:

"Verification of the Executable Object Code is primarily performed by testing. This can be partially assisted by a combination of model simulation and specific analysis … This combination can be used to partially satisfy the following software testing objectives".

Those objectives include the compliance of EOC with high-level and low-level requirements, test coverage of software structure, and data coupling and control coupling. Additional verification activities must be performed on the target hardware to fully satisfy these objectives. The document goes on to say that when certification credit is sought from model simulation to partially satisfy software testing objectives and test coverage regarding high-level requirements then it must be ensured that the same design model is used for code generation and to produce the EOC.

It also specifies that plans are required to define which requirements and associated test and test coverage activities are to be satisfied at the model level, and which are to be exercised on the target.

[4] Based on table MB.1.1 from RTCA DO-331. Copyright © 2011 RTCA, Inc. All rights acknowledged.

Verification on target

DO-331 §MB.6.8.2 goes on to say:

"… specific tests should still be performed in the target environment … The following software testing and test coverage objectives cannot be satisfied by the model simulation since simulation cases should be based on the requirements from which the model is developed."

These objectives listed include EOC robustness, its compliance with low-level requirements, and test coverage of low-level requirements.

The supplement then goes on to identify the various forms of verification objectives that can only be met on the target, including confirmation of compatibility with the target hardware, and hardware/software integration testing. It also lists various types of errors that can and cannot be revealed at the simulation level and can only be detected on the target hardware.

Finally, DO-331 §MB.B.11 (FAQ #11) addresses the questions of model coverage activity:

"Model coverage analysis is different than structural coverage analysis and therefore model coverage analysis does not eliminate the need to achieve the objectives of structural coverage analysis per DO-178C section 6.4.4.2."

It goes on to state that model coverage analysis can be considered in very specific scenarios, "…on a case-by-case basis and agreed upon by the certification authorities …"

As a result, most organizations do some of the verification activities within the model but then re-affirm the results of those activities on the target hardware to ensure that they meet the necessary criteria for meeting objectives.

The integration of test and modelling tools help to achieve that seamlessly, including the static analysis of generated code, the collection of code coverage from model execution, and the migration of model tests into an appropriate form for execution on the target hardware (Figure 26).

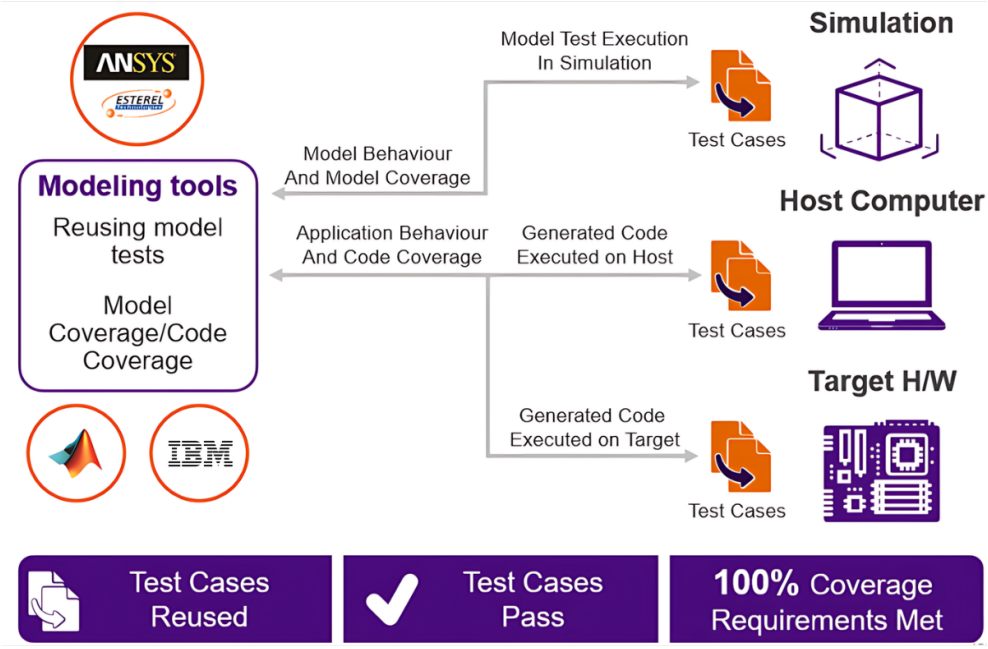


Figure 27: Migrating test cases from modelling tools to the LDRA tool suite for regression on target

DO-332 Object-Oriented Technology and Related Techniques Supplement

In the early 2000s, object-oriented technology was viewed in the commercial avionics space as novel and unproven. Around that time the Certification Authorities Software Team (CAST) published papers to enumerate concerns and limitations. These were called CAST 4 and CAST 8 and were published in 2000 and 2002 respectively.

As DO-178B was updated to DO-178C it was decided that these concerns, vulnerabilities, and subsequent additional objectives associated with object-oriented technologies would be addressed not by the original standard but rather a supplement, "DO-332 - Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A". This new supplement describes concepts and key features of object-oriented technologies and related techniques, discusses their impact on the planning, development, and verification processes, and enumerates their vulnerabilities.

OO Objectives

Two pertinent objectives are included in the DO-332 supplement:

- **§OO.6,7.1** Verify local type consistency
- **§OO.6.8.1** Verify the use of dynamic memory management is robust

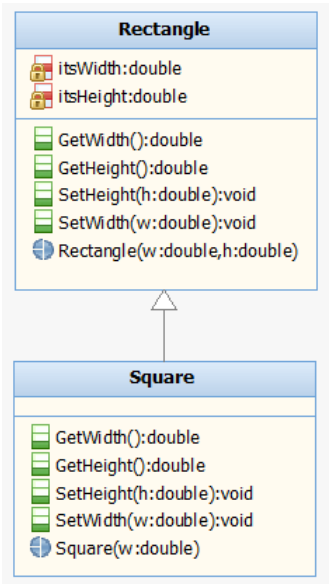
It is useful to understand Liskov's Substitution Principle in relation to the first of these.

"Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T"

This can be visualized by reference to an example. In common parlance, a square is a type of rectangle. The term "is a" suggests a relationship that could be represented through inheritance in programming.

It would seem reasonable to have SetWidth and SetHeight methods in a Rectangle class. But using SetWidth and SetHeight for a Rectangle object that is a Square necessitates ensuring that both dimensions are the same, leading to the possibility of incongruities.

Liskov Substitution Principle tests check for such problems. Consider (Figure 28)




```
virtual void Rectangle::SetWidth (double w)  {
      itsWidth=w;
}

    Post condition: itsHeight does not change

void Square::SetWidth (double w) {
     Rectangle::SetWidth(w);
     Rectangle::SetHeight(w);
}

     Post condition: itsHeight set to w
```

Figure 28: Parent and child class showing code that violates type consistency

In object-oriented languages, inheritance allows the behaviour of "superclasses" to be overridden by subclasses. As described in DO-332 FAQ #14, ensuring safe use of inheritance, method override, and dynamic dispatch is challenging as the nature of these techniques can make it unclear from a simple review which method is executed at any call point in a program. Overridden behaviour in instantiated subclasses may alter the behaviour beyond the intended scope of the superclass and violate type consistency.

As DO-332 §OO.6.7.1 further describes, this means that the preconditions of the parent class must not be strengthened, and the postconditions and invariants defined on the state of a class must not be weakened.

From a verification standpoint, DO-332 §OO.6.7.2 suggests that one of the following activities must be performed:

- Verify substitutability using formal methods.
- Ensure that each class passes all the tests of all its parent types which the class can replace
- For each call point, test every method that can be invoked at that call point (pessimistic testing)

The first of these applies to the small minority of development teams who are using formal methods, whilst the third (once commonly referred to as flattened class testing) requires that each possible dispatch is tested at every call point in a program. That can easily cause a combinatorial explosion of test cases, dramatically increasing the verification burden.

That leaves the second option as the most practical for most people - to ensure type consistency, without the burden of pessimistic testing. Doing so requires that each class and its methods must pass all tests of every superclass for which it can be substituted (DO 332 FAQ #34).

In the Rectangle and Square example, type consistency is violated. Reusing test cases from the parent class Rectangle on the subclass Square will highlight that (Figure 29).

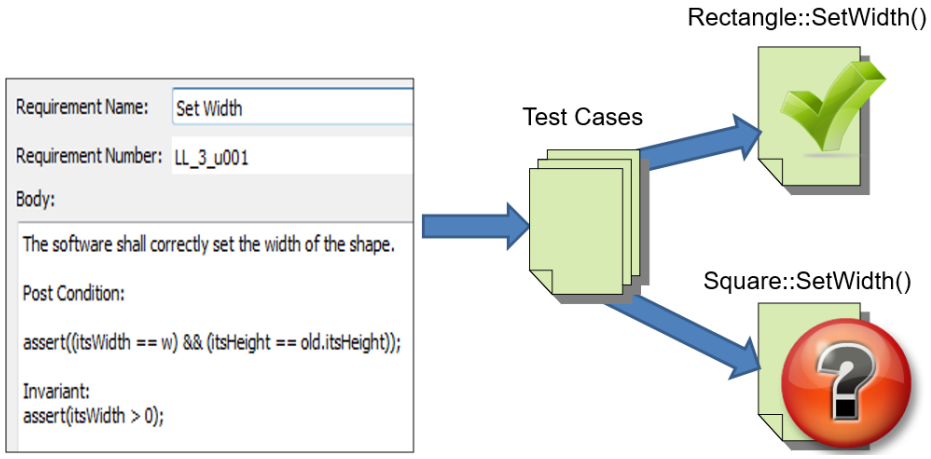


Figure 29: The Rectangle Class test cases are applied to its Square subclass to ensure local type consistency

A negative test case for the Rectangle class shows that as expected, setting the width has no impact on the height. When that same negative test is applied to the Square class's SetWidth method, one can clearly see that itsHeight is changed (Figure 30).

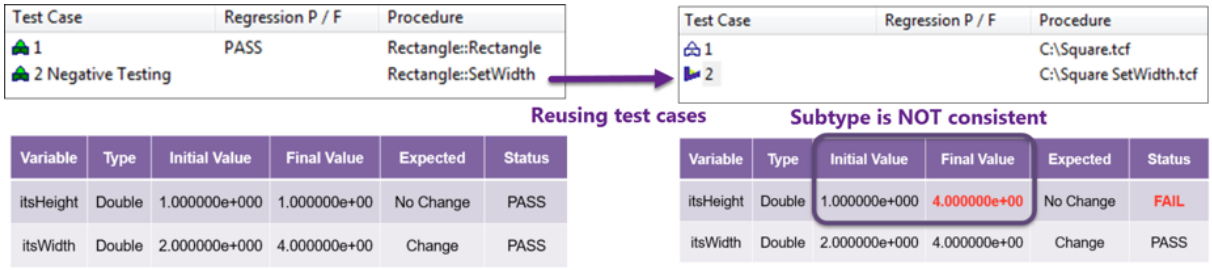


Figure 30: Parent class test cases being reused to test a subclass to detect an inconsistent subtype

A range of static and dynamic analysis techniques can be deployed in order to fulfil DO 332 A-7 §OO.6.8.1 "Verify the use of dynamic memory management is robust" and the related vulnerabilities outlined in Annex OO.D.1.6.1.

Tracking memory allocation and deallocation helps to ensure the proper freeing of memory, as do associated checks prior to dereferencing. Low-level testing provides a mechanism to explore various allocation/deallocation scenarios to help ensure that vulnerabilities described in §OO.D.1.6.1 are addressed. Timing hooks within the low-level tests help characterize allocation/deallocation timing and dynamic data flow analysis tracks references and updates of data elements in runtime to detect lost updates and stale references.

Other considerations

When using object-oriented technologies or related techniques, there are various other factors to consider:

- **Source to Object traceability:** As mentioned in §OO.D.1.2.1, source to object code traceability may be more difficult to correlate in object-oriented languages. OCV solutions provide a graphical comparison of assembly code coverage and high-order language coverage (i.e. C++) to ensure that the source coverage data accounts for variations in the structure of executable object code (EOC) as compared to the source code.
- **Traceability to child classes:** §OO.5.2.2.i states "Develop a locally type consistent class hierarchy with associated low-level requirements whenever substitution is relied upon". In other words, a requirement that traces to a method implemented in a class should also trace to the method in its subclasses when the method is overridden in that subclass (FAQ #9). Static analysis and code visualization exposes inheritance relationships within the analysed code, making traceability gaps across class hierarchies easier to detect and remedy.
- **Coding standard for object-oriented languages:** Languages such as C++ allow for tremendous syntactic/semantic flexibility. Standards such as MISRA C++ 2023 and JSF AV++ help quickly define a language subset and best practices to provide a baseline for software coding standards used in specific projects.
- **Challenges with structural coverage and low-level testing:** Structural coverage of destructors, instantiating complex data types for testing, testing templated classes and overloaded operators, and accessing private members, are just some of the challenges that arise when working with object-oriented technologies or related techniques. Tools need to be equipped to address these challenges and reduce the cost of verification, while preserving the integrity/credibility of the verification activities.

DO-333 Formal Methods Supplement to DO-178 and DO-278A

"DO-333 Formal Methods Supplement to DO-178 and DO-278A" provides guidance on the use of formal methods. Formal methods are mathematically based techniques for the specification, development, and verification of software and hardware systems. DO-333 integrates these methods into the existing framework of DO-178C to provide an alternative to the more established methods of providing a rigorous and reliable certification process.

The document outlines various formal methods tools, such as theorem provers, model checkers, and abstract interpretation tools, and provides criteria for their qualification. It details how these formal methods can be leveraged to verify complex systems, complementing the traditional approaches established by DO-178C. Where formal methods are leveraged as part of a safety case, DO-333 helps ensure that safety-critical systems meet the usual stringent reliability and performance standards.

Conclusions

The use of traceability, test management and static/dynamic analysis tools for an airborne software project that meets the DO-178C certification requirements offers significant productivity and cost benefits. Tools generally make compliance checking easier, less error prone and more cost effective. In addition, they make the creation, management, maintenance and documentation of requirements traceability straightforward and cost effective.

In particular, the provision of an automated, comprehensive software tool chain offering complete 'end-to-end' traceability across the development life cycle, encompassing requirements, code, on-target tests, artifacts, and objectives with both static and dynamic analysis capabilities is invaluable to developers and project managers alike.

Development in compliance with DO-178C will never be an easy task. However, the right tools can be very helpful in making such work as easy as it can possibly be.

Works cited

[1] RTCA, "RTCA," [Online]. Available: http://www.rtca.org/ . [Accessed 14th June 2024].

[2] EUROCAE, "EUROCAE," Userfull, 2024. [Online]. Available: https://eurocae.net/. [Accessed 14th June 2024].

[3] RTCA, Inc., "RTCA DO-178 - Software Considerations in Airborne Systems and Equipment Certification," RTCA, Inc., 18th November 1981. [Online]. Available: https://my.rtca.org/productdetails?id=a1B36000001IcmwEAC. [Accessed 14th June 2024].

[4] European Organization for Civil Aviation Equipment (EUROCAE), ED-12 - Software Considerations in Airborne Systems and Equipment Certification, Cologne, Germany: EUROCAE, 1985.

[5] RTCA, Inc., RTCA DO-178B - Software Considerations in Airborne Systems and Equipment Certification, RTCA, Inc., 1992.

[6] European Organization for Civil Aviation Equipment (EUROCAE), ED-12B - Software Considerations in Airborne Systems and Equipment Certification, Cologne, Germany: EUROCAE, 1992.

[7] RTCA, Inc., RTCA DO-178C - Software Considerations in Airborne Systems and Equipment Certification, RTCA, Inc., 2011.

[8] European Organization for Civil Aviation Equipment (EUROCAE), ED-12C - Software Considerations in Airborne Systems and Equipment Certification, Paris, France: EUROCAE, 2011.

[9] Federal Aviation Administation (FAA), "Federal Aviation Administation (FAA)," [Online]. Available: https://www.faa.gov/. [Accessed 3 November 2021].

[10] European Union Aviation Safety Agency (EASA), "European Union Aviation Safety Agency (EASA)," 2021. [Online]. Available: https://www.easa.europa.eu/. [Accessed 3 November 2021].

[11] Transport Canada Civil Aviation (TCCA), "Government of Canada - Aviation," TCCA, 13th December 2013. [Online]. Available: https://tc.canada.ca/en/aviation. [Accessed 30th June 2024].

[12] RTCA, Inc., RTCA DO-330 - Software Tool Qualification Considerations, Washington DC: RTCA, Inc., 2011.

[13] RTCA, Inc., RTCA DO-331 - Model-Based Development and Verification Supplement to DO-178C and DO-278A, Washington DC: RTCA, 2011.

[14] RTCA, Inc., RTCA DO-332 - Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A, Washington DC: RTCA, 2011.

[15] RTCA, Inc., RTCA DO-333 - Formal Methods Supplement to DO-178C and DO-278A, Washington DC: RTCA, Inc., 2011.

[16] LDRA, "LDRA tool suite," LDRA, [Online]. Available: https://ldra.com/products/ldra-tool-suite/. [Accessed 23 March 2022].

[17] RTCA, Inc., "RTCA DO-278 - Guidelines For Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance," RTCA, Inc., May 3rd 2002. [Online]. Available: https://my.rtca.org/productdetails?id=a1B36000001IciHEAS. [Accessed 15th June 2024].

[18] European Organization for Civil Aviation Equipment (EUROCAE), ED-109 - Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems, Paris, France: EUROCAE, 2003.

[19] Federal Aviation Administration (FAA), Advisory Circular 20-193: Use of Multi-Core Processors., Washington, DC.: FAA, 2024.

[20] Federal Aviation Administration (FAA), "AC 20-193 - Use of Multi-Core Processors," U.S. Derpartment of Transportation, 8th January 2024. [Online]. Available: https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentID/1036408. [Accessed 5th March 2024].

[21] RTCA, Inc., RTCA DO-278A - Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems, Washington DC: RTCA, Inc., 2011.

[22] SAE International, "ARP4754B - Guidelines for Development of Civil Aircraft and Systems," 20th December 2023. [Online]. Available: https://www.sae.org/standards/content/arp4754b/. [Accessed 14th June 2024].

[23] SAE International, "ARP4761A - Guidelines for Conducting the Safety Assessment Process on Civil Aircraft, Systems, and Equipment," 20th December 2023. [Online]. Available: https://www.sae.org/standards/content/arp4761a/. [Accessed 14th June 2024].

[24] RTCA, Inc., DO-254 - Design Assurance Guidance for Airborne Electronic Hardware, Washington, DC: RTCA, Inc., 2008.
[25] European Organization for Civil Aviation Equipment (EUROCAE), ED-80 - Design Assurance Guidance for Airborne Electronic Hardware, Paris, France: EUROCAE, 2000.
[26] RTCA, Inc., DO-160G - Environmental conditions and test procedures for airborne equipment, Washington, DC: RTCA, Inc., 2010.
[27] European Organization for Civil Aviation Equipment (EUROCAE), ED-14G - Environmental Conditions and Test Procedures for Airborne Equipment, Paris, France: EUROCAE, 2010.
[28] RTCA, Inc., "DO-297 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations," 8 November 2005. [Online]. Available: https://my.rtca.org/NC__Product?id=a1B36000001IchGEAS. [Accessed 6 January 2022].
[29] European Organization for Civil Aviation Equipment (EUROCAE), ED-124 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, Paris, France: EUROCAE, 2010.
[30] RTCA, Inc., DO-326A Airworthiness Security Process Specification, Washington, DC: RTCA, Inc., 2014.
[31] European Organisation for Civil Aviation Equipment (EUROCAE), ED-202A - Airworthiness Security Process Specification, Paris, France: EUROCAE, 2014.
[32] RTCA, Inc., DO-356 - Airworthiness Security Methods and Considerations, RTCA, 2018.
[33] European Organisation for Civil Aviation Equipment (EUROCAE), ED 203 - Airworthiness Security Methods and Considerations, EUROCAE, 2018.
[34] European Organization for Civil Aviation Equipment (EUROCAE), ED-215 - Software Tool Qualification Considerations, Paris, France: EUROCAE, 2012.
[35] European Organization for Civil Aviation Equipment (EUROCAE), ED-216: Guidelines for the Use of Modelling Techniques and Tools for Airborne Systems Supplement to ED-12C and ED-109A., Paris, France: EUROCAE, 2017.
[36] European Organization for Civil Aviation Equipment (EUROCAE), ED-217 - Object-Oriented Technology and Related Techniques Supplement to ED-12C and ED-109A., Paris, France: EUROCAE, 2017.
[37] European Organization for Civil Aviation Equipment (EUROCAE), ED-218 - Formal Methods Supplement to ED-12C and ED-109A., Paris, France: EUROCAE, 2018.
[38] Federal Aviation Administration, "AC 20-148 - Reusable Software Components Document Information," 7 December 2004. [Online]. Available: https://www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentID/22207. [Accessed 22 September 2021].
[39] European Union Aviation Safety Agency (EASA), "3.1 Use of ED-124," in AMC-20 - General Acceptable Means of Compliance for Airworthiness of Products, Parts and Appliances, EASA, 2020, p. 590.
[40] LDRA, "LDRA Certification Services (LCS)," [Online]. Available: https://ldra.com/certification-services/. [Accessed 25th June 2024].
[41] LDRA, "Data Sheet - LDRA Compliance Management System (LCMS)," May 2024. [Online]. Available: https://ldra.com/documentviewer.php?file=LCMS-Data-Sheet-v2.1.pdf. [Accessed 25th June 2024].
[42] Microsoft, "Microsoft Word," [Online]. Available: https://www.microsoft.com/en-gb/microsoft-365/word. [Accessed 20 January 2021].
[43] IBM, "IBM Engineering Requirements Management DOORS Family," [Online]. Available: https://www.ibm.com/uk-en/products/requirements-management. [Accessed 20 January 2021].
[44] Siemens, "Polarion ALM," [Online]. Available: https://polarion.plm.automation.siemens.com/products/polarion-alm. [Accessed 20 January 2021].
[45] Object Management Group(R) , "Requirements Interchange Format," Object Management Group(R), 2021. [Online]. Available: http://www.omg.org/spec/ReqIF/ . [Accessed 10 August 2021].
[46] LDRA, "TBrun®," LDRA, [Online]. Available: https://ldra.com/products/tbrun/. [Accessed 23 March 2022].
[47] Certification Authorities Software Team (CAST), Position Paper CAST-32A - Multicore processors, CAST, 2016.
[48] LDRA, "TBwcet®," [Online]. Available: https://ldra.com/products/tbwcet/. [Accessed 3rd February

2023].
[49] C. King, “Ubuntu wiki: stress-ng,,” 7th October 3030. [Online]. Available: https://wiki.ubuntu.com/Kernel/Reference/stress-ng. [Accessed 3rd February 2023].
[50] D. Iorga, T. Sorensen, J. Wickerson and A. F. Donaldson, “Slow and Steady: Measuring and Tuning Multicore Interference,” in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Syndey, Australia, 2020.
[51] The MathWorks, Inc., “MathWorks Simulink,” The MathWorks, Inc., 1994-2022. [Online]. Available: https://uk.mathworks.com/products/simulink.html. [Accessed 28 April 2022].
[52] IBM, “IBM Engineering Systems Design Rhapsody - Developer,” IBM, [Online]. Available: https://www.ibm.com/uk-en/products/uml-tools/details. [Accessed 28 April 2022].
[53] ANSYS, Inc, “Ansys SCADE Suite,” ANSYS, 2021. [Online]. Available: https://www.ansys.com/products/embedded-software/ansys-scade-suite. [Accessed 10 August 2021].
[54] The MISRA Consortium, MISRA AC:2023 - Guidelines for the use of language subsets for automatic code generation purposes, United Kingdom: The MISRA Consortium Limited, 2023.
[55] Carnegie Mellon university Software Engineering Institute, “SEI CERT C++ Coding Standard,” Carnegie Mellon university Software Engineering Institute, 29 May 2020. [Online]. Available: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682. [Accessed 6 January 2022].
[56] Carnegie Mellon University Software Engineering Institute, “SEI CERT C Coding Standard,” Carnegie Mellon University Software Engineering Institute, 5 December 2018. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c. [Accessed 6 January 2022].
[57] Homeland Security Systems Engineering and Development Institute, “CWE Common Weakness Enumeration,” The MITRE corporation, August 2021. [Online]. Available: https://cwe.mitre.org/. [Accessed 24 September 2021].
[58] The MISRA Consortium, MISRA C:2023: Guidelines for the use of the C language in critical systems, United Kingdom: The MISRA Consortium Limited, 2023.
[59] The MISRA Consortium, MISRA C++:2023 - Guidelines for the use of C++17 in critical systems, United Kingdom: The MISRA Consortium Limited, 2023.
[60] Lockheed Martin, JSF AV C++ Coding Standards, Fort Worth, Texas: Lockheed Martin Corporation, 2005.
[61] High Integrity C++ Working Group, High Integrity C++ Coding Standard v4.0, PRQA Programming Research, part of Perforce Software., 2013.
[62] International Real-Time Ada Workshops., Ravenscar Profile, Ravenscar, North Yorkshire, England: Ada Resource Association, 2012 (Ada 2012 standard).
[63] AdaCore and Altran UK Ltd., Spark 2014 User’s Guide, Samurai Media Limited, 2017.
[64] Federal Aviation Administration (FAA), Advisory Circular 20-170A: Integrated Modular Avionics (IMA) Development, Verification, Validation, Integration, and Approval Using DO-297 and TSO C153A., Washington, DC: FAA, 2021.
[65] EASA, “AMC 20-193 Use of multi-core processors,” 2022. [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/annex_i_to_ed_decision_2022-001-r_amc_20-193_use_of_multi-core_processors_mcps.pdf. [Accessed 18th January 2023].

Appendices

Appendix A – LDRA tools and DO-178C table A-6

DO-178C - Table A-6 - Verification of Outputs of the Software Design Process.				
DO-178C Objective	Associated Activity	Do LDRA tools facilitate this activity? (Yes/No)	If Yes, credit given by the tool (Partial/ Full)	Details of the credit given
(1) Executable Object Code complies with high-level requirements	6.4.2	Yes	Full	LDRA tools provide a framework for the implementation, execution and traceability of requirements-based tests, of both normal range and robustness types.
	6.4.2.1	Yes	Full	See 6.4.2 above
	6.4.3	Yes	Full	See 6.4.2 above
	6.5	Yes	Full	LDRA tools provide bidirectional traceability between requirements, source code, associated tests and test outcomes.
(2) Executable Object Code is robust with high-level requirements	6.4.2	Yes	Full	See 6.4.2 above
	6.4.2.2	Yes	Full	See 6.4.2 above
	6.4.3	Yes	Full	See 6.4.2 above
	6.5	Yes	Full	LDRA tools provide bidirectional traceability between requirements, source code, associated tests and test outcomes.
(3) Executable Object Code complies with low-level requirements				
	6.4.2	Yes	Full	See 6.4.2 above
	6.4.2.1	Yes	Full	See 6.4.2 above
	6.4.3	Yes	Full	See 6.4.2 above
	6.5	Yes	Full	LDRA tools provide bidirectional traceability between requirements, source code, associated tests and test outcomes.
(4) Executable Object Code is robust with low-level requirements.				
	6.4.2	Yes	Full	See 6.4.2 above
	6.4.2.2	Yes	Full	See 6.4.2 above
	6.4.3	Yes	Full	See 6.4.2 above
	6.5	Yes	Full	LDRA tools provide bidirectional traceability between requirements, source code, associated tests and test outcomes.
(5) Executable Object Code is compatible with target compiler.	6.4e	Yes	Full	The source to object code traceability features of the LDRA tool suite have the capacity to demonstrate a valid relationship between source and object code.

Appendix B – LDRA tools and DO-178C table A-7

DO-178C - Table A-7 - Verification of Outputs of the Software Coding & Integration Processes.				
DO-178C Objective	Associated Activity	Do LDRA tools facilitate this activity? (Yes/No)	If Yes, credit given by the tool (Partial/ Full)	Details of the credit given
(5) Test coverage of software structure (modified condition/ decision coverage) is achieved.	6.4.4.2.a	Yes	Full	LDRA structural coverage analysis measures MC/ DC
	6.4.4.2.b	Yes	Full	LDRA structural coverage analysis measures MC/ DC on the Source Code executed in the native target execution environment. LDRA also offers test coverage tools for Source to Object Code traceability.
	6.4.4.2.d	Yes	Partial	LDRA static analysis tools can identify dead code. LDRA structural coverage analysis facilities identify untested source and object code. LDRA requirements traceability tools aid the identification of missing requirements and missing code.
	6.4.4.3	Yes	Partial	See 6.4.4.2.d above.
(6) Test coverage of software structure (decision coverage) is achieved.	6.4.4.2.a	Yes	Full	LDRA structural coverage analysis measures decision coverage
	6.4.4.2.b	Yes	Full	LDRA also offers test coverage tools for Source to Object Code traceability.
	6.4.4.2.d	Yes	Full	LDRA static analysis tools can identify dead code. LDRA structural coverage analysis facilities identify untested source and object code. LDRA requirements traceability tools aid the identification of missing requirements and missing code.
	6.4.4.3			See 6.4.4.2.d above.
(7) Test coverage of software structure (statement coverage) is achieved.	6.4.4.2.a	Yes	Full	LDRA structural coverage analysis measures statement coverage
	6.4.4.2.b	Yes	Full	LDRA also offers test coverage tools for Source to Object Code traceability.
	6.4.4.2.d	Yes	Full	LDRA static analysis tools can identify dead code. LDRA structural coverage analysis facilities identify untested source and object code. LDRA requirements traceability tools aid the identification of missing requirements and missing code.
	6.4.4.3			See 6.4.4.2.d above.

DO-178C - Table A-7 - Verification of Outputs of the Software Coding & Integration Processes.				
DO-178C Objective	Associated Activity	Do LDRA tools facilitate this activity? (Yes/No)	If Yes, credit given by the tool (Partial/ Full)	Details of the credit given
(8) Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.2.c	Yes	Full	The data and control coupling features of the LDRA tool suite support the analysis of explicit data and control flow between software components and applications.
	6.4.4.2.d		Full	LDRA static analysis tools can identify dead code. LDRA structural coverage analysis facilities identify untested source and object code. LDRA requirements traceability tools aid the identification of missing requirements and missing code.
	6.4.4.3			See 6.4.4.2.d above.
(9) Verification of additional code, that cannot be traced to Source Code, is achieved	6.4.4.2.b	Yes	Full	The source to object code traceability features of the LDRA tool suite provides an Object Code Verification (OCV) solution that includes source code to object code traceability and object code coverage analysis.

FM 26376

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
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

DO-178C white paper v3.0 08/24