



Whitepaper

DO-178C demystified: Strategies for efficient certification



Background

If you're working towards [DO-178C certification](#), you're probably juggling far more than software development. Traceability, testing, structural coverage, tool qualification, multicore processors, and certification evidence all demand time, effort, and expertise. The good news is that DO-178C is not as daunting as it can sometimes appear - and once you understand the process and the objectives behind it, certification becomes far more manageable. This guide explains the key concepts behind DO-178C, explores the technologies and workflows used on successful projects, and shows how modern verification, target analysis, debugging, and trace technologies can help reduce effort and risk.

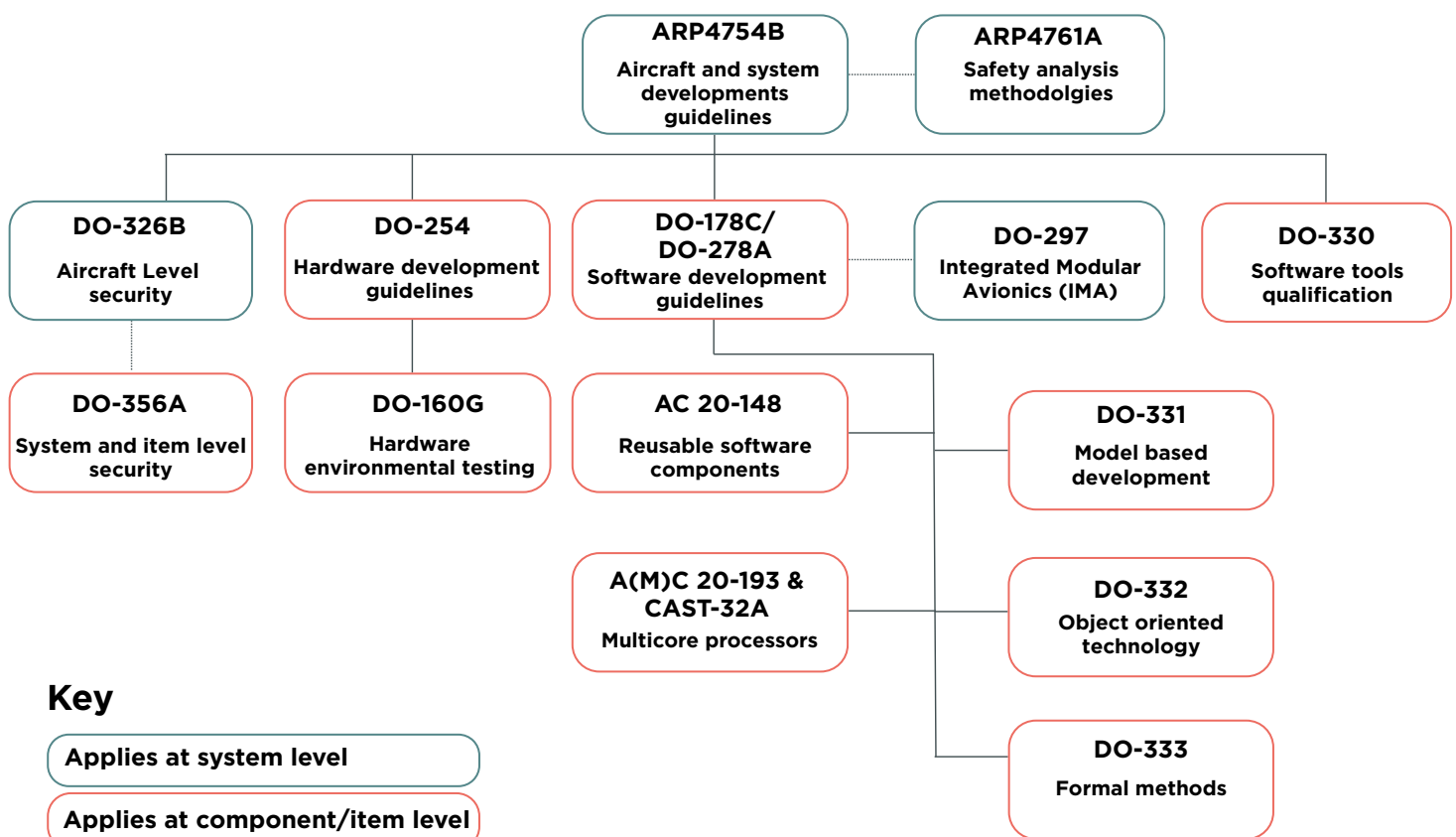
Before diving into the details, it's worth understanding where DO-178C fits within the broader airborne systems development landscape.

DO-178C in context

DO-178C does not exist in isolation. It forms part of a broader framework of aerospace standards and guidance covering aircraft systems, safety assessment, hardware, and software. Although technically guidance rather than a standard, DO-178C is generally treated as one by the aerospace community - as will we.

At the system level, [ARP4754B](#) defines the processes used to develop and certify aircraft systems, while [ARP4761A](#) provides guidance on safety assessment and failure analysis. Together, they determine the level of assurance required for the hardware and software that implement those systems.

Within that framework, DO-178C provides guidance for airborne software development and verification, while DO-254 addresses airborne electronic hardware. Other standards and guidance documents address specific technologies and challenges, including integrated modular avionics (IMA), multicore processors, model-based development, object-oriented technologies, and formal methods.



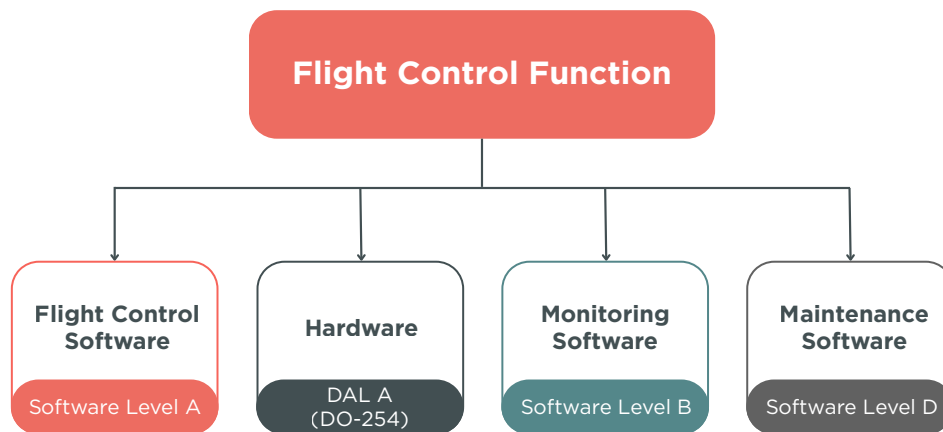
ARP4754B, DALs, and software levels

Before development begins, ARP4754B requires safety assessments to determine the potential consequences of system failures. The resulting failure classifications drive the assignment of Development Assurance Levels (DALs) to the affected systems.

Those system-level DALs are then decomposed into hardware DALs and Software Levels for the contributing subsystems, with DO-178C defining the objectives that apply to each software level.

Importantly, software levels do not necessarily correspond directly to system DALs. The level assigned to a software component depends on the contribution that software makes to the system-level failure condition being considered.

Failure condition	System DAL	Software Level
Catastrophic	A	A
Hazardous	B	B
Major	C	C
Minor	D	D
No effect	E	E



DO-178C Certification objectives

One of the biggest challenges in any DO-178C project is demonstrating that the development and verification activities have been completed correctly and consistently. To achieve this, the standard defines a series of planning, development, verification, and quality assurance objectives that must be satisfied throughout the software life cycle.

You demonstrate compliance progressively rather than at the end of the project. Certification authorities therefore conduct a series of Stage of Involvement (SOI) reviews to assess plans, processes, development artifacts, verification activities, and certification evidence as they evolve.

Ultimately, successful certification depends upon the quality and completeness of the evidence produced throughout the project. This evidence is brought together within the Software Accomplishment Summary (SAS), which provides a consolidated view of the software life cycle, verification results, tool qualification activities, and compliance status. A clear, well-structured body of evidence can significantly reduce certification effort and risk, while helping to avoid costly surprises late in the program.

Got questions as you read?

Our aerospace verification engineers have spent 20+ years working DO-178C programs from DAL D to DAL A. If something in here maps to a challenge you're facing.

Book a meeting with our [DO-178C specialist →](#)



Compliance management

Requirements must be captured, implemented, verified, and linked to corresponding code and test cases. Configuration management adds further complexity, as software artifacts evolve throughout the project life cycle and multiple versions of requirements, designs, code, tests, and certification evidence must remain aligned. Gaps in traceability, inconsistent processes, disconnected tools, and poorly managed changes can all introduce risk, increase verification effort, and complicate certification activities.

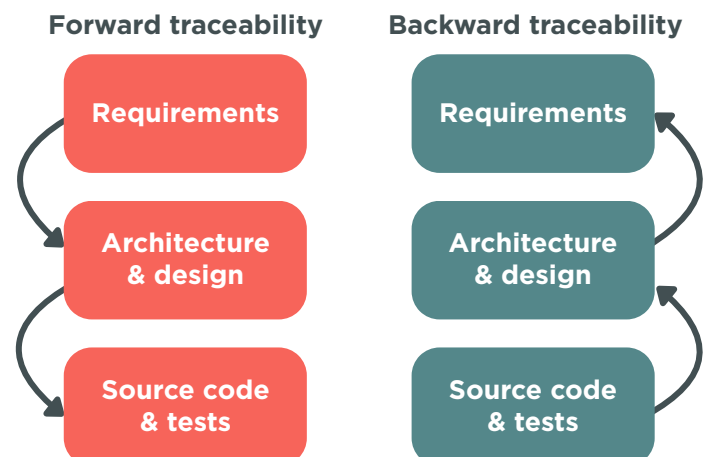
Successful projects maintain visibility and control throughout the software life cycle. Requirements, development activities, verification results, reviews, configuration data, and certification artifacts must all work together to create a coherent body of evidence. In practice, this means establishing seamless connections between the people, processes, and technologies involved in demonstrating compliance.

DO-178C Software development and verification processes

Key elements of the DO-178C software life cycle include the practices of traceability and structural coverage analysis. You need bidirectional traceability across the life cycle.

Structural coverage analysis (including code coverage, and data coupling and control coupling) quantifies the extent to which requirements-based testing has exercised the software. Together, these practices ensure that code has been implemented to address every system requirement and that the implemented code has been verified.

Software tools offer significant benefits throughout the software development and verification processes defined in §5.0 and §6.0.



DO-178C Section 5.0: Software development processes

DO-178C §5.0 defines five software development processes: the 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).

You need a development framework before software development can begin. Requirements must be defined and decomposed, software architectures created, programming languages selected, coding standards adopted, development tools chosen, and a traceability process established to maintain relationships between development artifacts.

For example, §5.3 defines objectives for the software coding process, including the implementation of low-level requirements and adherence to software coding standards. Static analysis tools automate many aspects of code review by comparing source code against the rules of the chosen coding standard, identifying defects, and highlighting potential compliance issues early in the development life cycle.

Similarly, §5.5 requires traceability to be established and maintained throughout the software development process. In practice, requirements evolve, designs mature, defects are corrected, and integration activities uncover new issues. Every change has the potential to affect requirements, source code, tests, reviews, and certification evidence.

“**Having a tool that will find 99% of coding standard violations in seconds is great**”
-Stephen Spence, Richland Technologies

Requirements traceability, coding standards compliance, static analysis, change impact analysis, and certification evidence management all help maintain visibility and control as projects grow in size and complexity. LDRA tools support these activities by automating many of them, helping maintain alignment between requirements, source code, reviews, tests, coverage results, and certification artifacts while reducing manual effort and the potential for error.

The outputs of the software development processes become the inputs to the software verification processes discussed in §6.0. Maintaining traceability between the two is therefore fundamental to demonstrating compliance with DO-178C.

High level
requirements

Low level
requirements

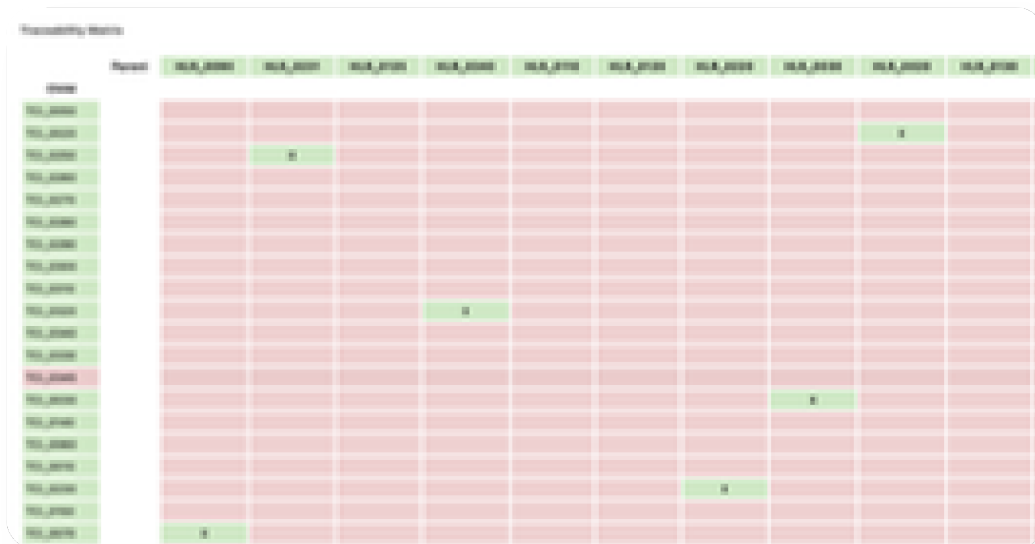
Source code

DO-178C Section 6.0: SOFTWARE VERIFICATION PROCESSES

§6.0 defines the software verification processes that provide objective evidence that the software satisfies its requirements. Verification is performed throughout the software life cycle using requirements-based testing and structural coverage analysis.

Dynamic analysis executes test cases and procedures against the Executable Object Code (EOC) on a representative target. Verification activities include low-level, software integration, and system testing, providing evidence that requirements have been correctly implemented while contributing to the structural coverage objectives defined by DO-178C.

As software evolves, maintaining the relationships between requirements, source code, test cases, coverage results, and certification evidence becomes increasingly challenging. LDRA tools help automate requirements-based testing, traceability, regression testing, change impact analysis, and certification evidence management, reducing manual effort while helping ensure that verification remains complete as projects evolve.



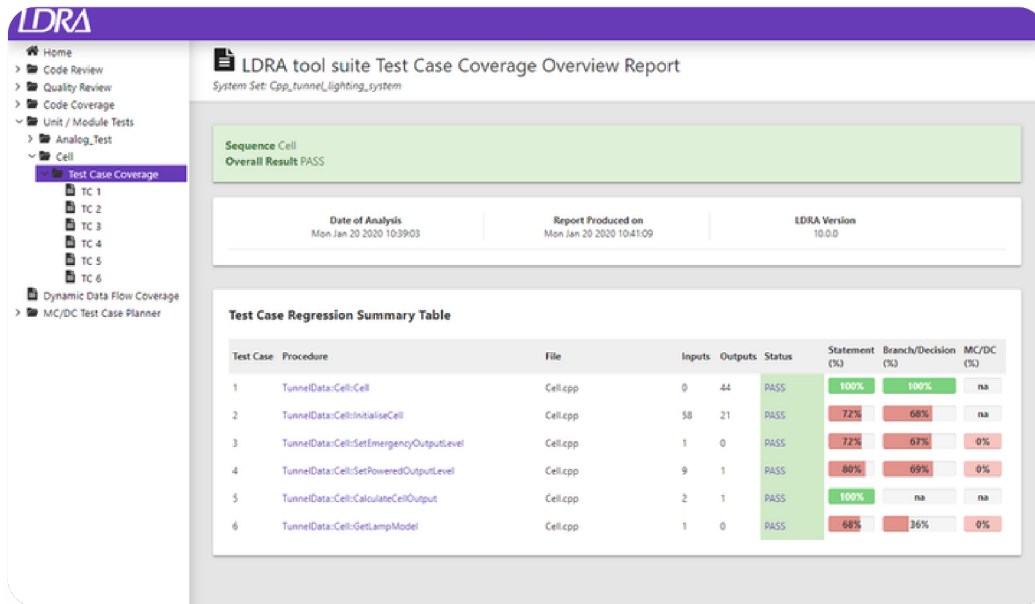
Structural coverage analysis objectives

Combined with traceability from requirements through to source code and test cases, structural coverage analysis provides objective evidence that system requirements have been correctly decomposed, implemented, and verified.

DO-178C §6.4.4 specifies the structural coverage objectives to be achieved according to Software Level, including statement, decision, and Modified Condition/Decision Coverage (MC/DC). Traditionally, structural coverage metrics have been collected by instrumenting a copy of the source code and executing the resulting application using requirements-based test cases. Structural coverage analysis then measures how much of the code has been exercised, guiding an iterative review and verification cycle until the coverage objectives have been achieved.

Increasingly, hardware trace technology provides a complementary approach during software development by deriving structural coverage from execution on the target, without modifying the application under test. This allows engineers to identify coverage gaps quickly and obtain rapid feedback as software evolves, reducing the need for repeated instrumented test cycles during day-to-day development.

Instrumentation remains the established route for generating DO-178C certification evidence, supporting activities such as MC/DC analysis and any additional structural coverage needed as projects iterate towards their coverage objectives. Together, hardware trace and instrumentation allow you to optimize the verification process while continuing to generate the evidence required for certification.



“

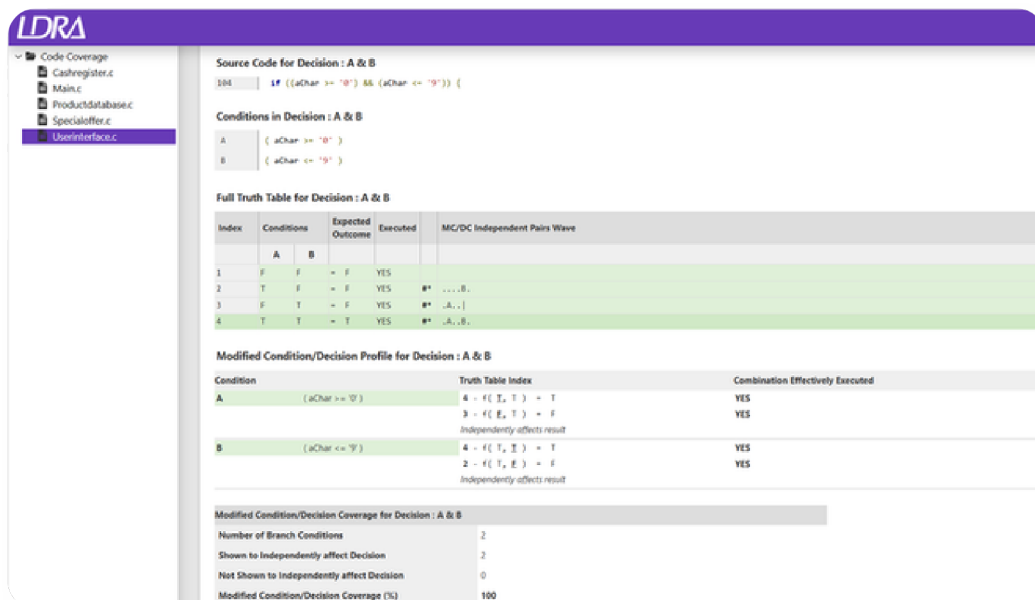
“The reduction in manual effort included the automatic production of complex test harnesses and stub code by the LDRA unit testing tool, which we would otherwise have had to write manually.”
- Tom Roberts, Ultra Datal

Modified Condition/Decision Coverage (MC/DC)

For Level A software, DO-178C requires Modified Condition/Decision Coverage (MC/DC) in addition to statement and decision coverage. MC/DC demonstrates that each condition within a decision can independently affect that decision's outcome.

MC/DC is a source-level coverage objective and is demonstrated using qualified instrumentation. It does not require every possible combination of conditions to be tested. Instead, it requires a carefully selected set of test cases that demonstrate the independent effect of each condition. For example, a decision containing six conditions has 64 possible combinations, yet only seven carefully chosen tests are needed to achieve MC/DC.

Selecting those test cases is not always straightforward. MC/DC planning tools help identify suitable input values, reducing manual effort while ensuring the required coverage objectives can be achieved efficiently.

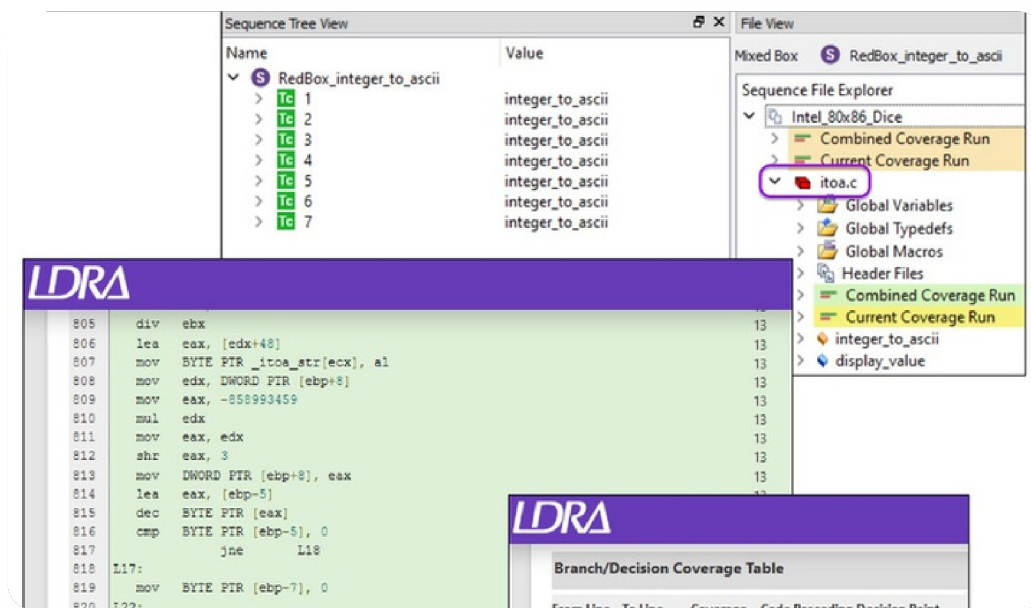


Source to object code traceability

For Level A software, demonstrating structural coverage at the source level alone is not sufficient. Compiler optimizations, code generation, and linker processing may produce object code that cannot be directly traced to individual source statements. To ensure that software behavior has not been compromised, DO-178C §6.4.4.2.b requires additional verification whenever a compiler, linker, or other tool generates code that cannot be directly traced to the source.

One approach is to compare the source code with the corresponding assembly code, allowing the generated object code to be reviewed in the context of the original implementation.

Object Code Verification (OCV) provides this additional confidence by measuring structural coverage at both the source and object code levels using qualified instrumentation.



Data Coupling and Control Coupling

DO-178C §6.4.4.2.c requires analysis to confirm that requirements-based testing has exercised the data and control coupling between software components. The resulting evidence must then be reviewed against the system requirements and software architecture.

Traditionally, data coupling and control coupling analysis can be tedious and time consuming. LDRA tools combine static and dynamic analysis to automate much of the process, reducing manual effort and helping eliminate errors.

Control coupling describes how one software component influences the execution of another. Analysis identifies the number and location of intra-file, file-to-file, and module-to-module calls, highlighting any gaps and guiding targeted verification activities.

Data coupling describes the dependence of one software component on data that is not exclusively under its control. Analysis focuses on how data elements are set and used across software component boundaries, providing evidence that requirements-based testing has exercised those interfaces correctly.

Execution time

DO-178C requires software execution times to be predictable and consistent with system requirements. The objective is to establish a Worst-Case Execution Time (WCET) for each task, considering both the target hardware and the software running on it.

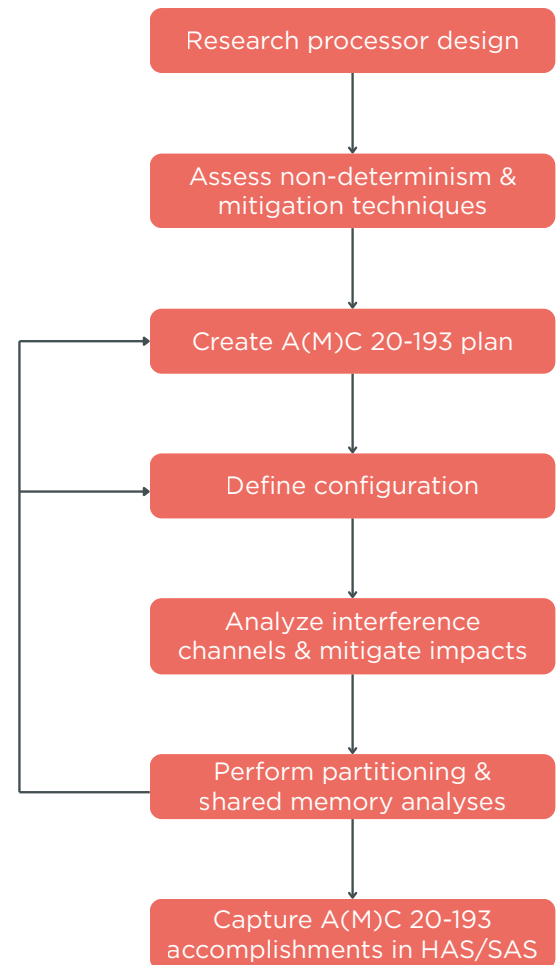
This becomes increasingly challenging as multicore processors become more common in avionics applications. Multiple processor cores often share hardware resources such as memory, caches, and interconnects, introducing interference effects that can increase execution times and reduce determinism.

FAA AC 20-193 and EASA AMC 20-193, collectively known as A(M)C 20-193, provide guidance for demonstrating that multicore interference has been adequately understood, mitigated, and verified.

Partitioning and other mitigation techniques can significantly reduce interference, but they rarely eliminate it entirely. Ultimately, you need to demonstrate that those mitigations are effective, and that can only be achieved through measurement.

The resulting interference research process is inherently iterative. Configurations are refined, mitigation strategies are evaluated, and measurements are repeated until the required behavior is achieved. Modern trace technology provides rapid visibility into execution behavior, interference channels, and the effectiveness of mitigation strategies, allowing configurations to be refined quickly during development.

Qualified execution time analysis tools then provide evidence in support of compliance activities and the production of Hardware and Software Accomplishment Summaries (HAS and SAS).



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.

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

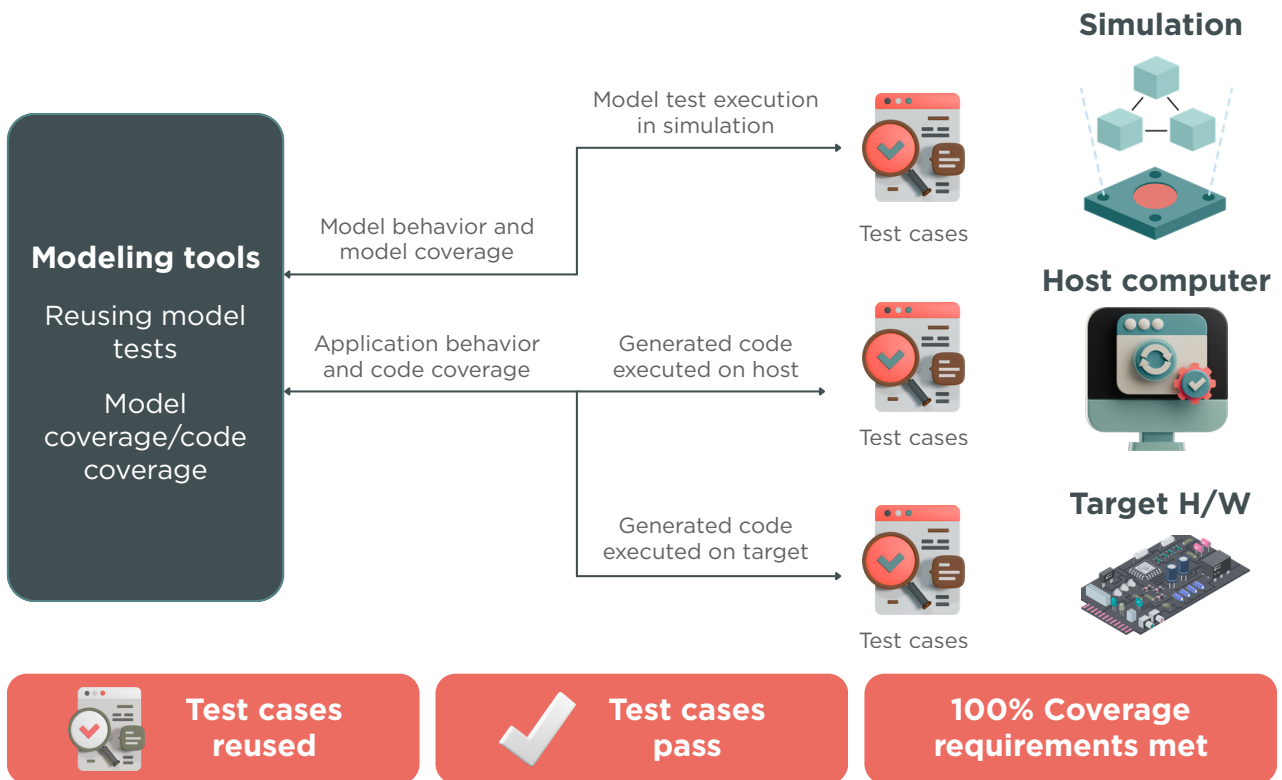
DO-330 Software Tool Qualification

DO-330 defines the framework used to establish confidence in software tools used within certification activities. Tool qualification remains a project-specific responsibility, but Tool Qualification Support Packages (TQSPs) can significantly reduce the time, cost, and risk associated with qualification activities.

DO-331 Model-Based Development

DO-331 addresses both Model-Based Systems Engineering (MBSE) and Model-Based Development (MBD), allowing verification activities to be performed at both model and target level. It defines which verification objectives may be partially satisfied through model simulation, and which continue to require verification on the target hardware.

The integration of modeling and verification tools helps support that workflow by enabling the analysis of generated code, the collection of coverage data, and the migration of model tests into an appropriate form for execution on the target hardware.



DO-332 Object-Oriented Technology

DO-332 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 objectives are particularly significant. The first is verification of local type consistency (§OO.6.7.1), often demonstrated through tests based on the Liskov Substitution Principle to show that derived classes correctly satisfy the behavior expected of their parent types.

The second is demonstrating the robust use of dynamic memory allocation and deallocation (§OO.6.8.1) through a combination of static analysis, dynamic analysis, and target testing.

DO-333 Formal Methods

DO-333 provides guidance for projects that choose to incorporate mathematically based formal methods into their certification strategy, either as an alternative or a complement to traditional verification techniques.

Conclusions

DO-178C compliance is inherently demanding. Requirements evolve, software changes, tests grow, and certification evidence accumulates rapidly throughout the life cycle.

Traceability, static analysis, dynamic analysis, testing, and certification evidence management tools help you maintain control of that complexity. They reduce manual effort, improve consistency, and make compliance activities easier to repeat, review, and justify.

Perhaps more importantly, they give you the visibility needed to make informed engineering decisions throughout the project. Requirements traceability, structural coverage, timing analysis, multicore interference research, and certification evidence all become easier to understand and manage when supported by an integrated tool chain.

No tools can make DO-178C compliance easy. The right tools can, however, make it significantly more efficient, predictable, and cost effective.

Ready to put this into practice?

TASKING solutions are qualified, integrated, and built for exactly the programs described in this paper - from structural coverage and MC/DC analysis to DO-330 TQS packages and SOI evidence generation. One toolchain, one vendor, one less risk to manage.

To see how the
TASKING toolchain supports DO-178C, [contact us](#) →