Developing DO-178B/C Compliant Software for Airborne Systems

INTRODUCTION

The investment required to produce software for airborne systems warrants scrupulous adherence to industry standards, such as DO-178B/C. The FAA uses DO-178B, formally titled Software Considerations in Airborne Systems and Equipment Certification, as a guide for determining software safety. That is, if the software has been developed according to DO-178B or DO-178C (the latest version of the standard), then the FAA will deem the software component airworthy.

In this paper, we'll discuss the following topics:

- Background about DO-178B/C
- Key differences between the versions
- Why policy-driven development is central to DO-178B/C compliance
- How Parasoft DTP for C and C++software helps organizations achieve DO-178B/C compliance

BACKGROUND ON DO-178B/C

The purpose of DO-178B/C is to provide guidance for software engineers so that they can ensure the airworthiness of the systems they develop. The FAA's airworthiness requirements determine whether or not the agency will issue their approval document called a TSO, or technical standard order. The standard is not explicitly mandated, but it is a critical component of the TSO process, thus a requirement for gaining FAA approval.

In the early days of aviation, safety considerations were overshadowed by the sheer achievement of human progress the new technology represented. But as the need for faster travel over greater distances with larger cargo loads emerged, so did the need for safety regulations. In a relatively short period of time, electronic interfaces replaced mechanical controls, elevating the need for industry-wide guidance for ensuring the safety of these systems. The standard was produced in response to a need for some sort of guide that vendors could use to gain FAA approval.

Those with experience developing embedded software systems that complies with industry standards will no doubt recognize and appreciate the qualifying language used in DO-178B/C. RTCA, the not-for-profit consortium that wrote the standard, specifically states that the organization's recommendations "may not be regarded as statements of official government policy" nor are they mandated by law. Instead, DO-178B/C" represents a consensus of the aviation community" in regard to software development.

PARASOFT®
Automated Software Testing

That is, DO-178B/C is *descriptive,* rather than *prescriptive*. The standard describes the goals of the processes, rather than prescribes the methods by which they are achieved. Consider 6.3.6 of DO-178B for example:

> **REVIEWS AND ANALYSIS OF THE TEST CASES, PROCEDURES AND RESULTS**
>
> The objective of these reviews and analyses is to ensure that the testing of the code was developed and performed accurately and completely. The topics should include:
>
> **a.** **Test cases:** The verification of test cases is presented in paragraph 6.4.4*.
>
> **b.** **Test procedures:** The objective is to verify that the test cases were accurately developed into test procedures and expected results.
>
> **c.** **Test results:** The objective is to ensure that the test results are correct and that discrepancies between actual and expected results are explained.
>
> ***6.4.4 states:** *Test coverage analysis is a two-step process, involving requirements-based coverage analysis and structural coverage analysis. The first step analyzes the test cases in relation to the software requirements to confirm that the selected test cases satisfy the specified criteria. The second step confirms that the requirements-based test procedures exercised the code structure. Structural coverage analysis may not satisfy the specified criteria. Additional guidelines are provided for resolution of such situations as dead code (subparagraph 6.4.4.3).*

The standard clearly describes the necessary verification activities and goals, but it does not prescribe methods for achieving those goals.

DIFFERENCES BETWEEN DO-178B AND DO-178C

In most respects, DO-178B and DO-178C are identical. Much of the language has been carried over; some of the broader goals, process and definitions have been explained in greater detail in DO-178C. For example, in chapter 6.1, which defines the purpose for the software verification process, DO-178C adds the following purpose with regard to the Executable Object Code:

- The Executable Object Code is robust with respect to the software requirements such that it can respond correctly to abnormal inputs and conditions.

This is in addition to the statement regarding the verification of the Executable Object Code defined in DO-178B:

- The Executable Object Code satisfies the software requirements (that is, intended function), and provides confidence in the absence of unintended functionality.

The additional requirement further defines the role of the Executable Object Code to ensure the safe functionality of the system, whereas the previous minimum scope of the verification process may lead to system failure. There are many instances where the language in DO-178C seeks to define concepts, functions, and process in more detail, but an exhaustive list of such updates are outside the scope of this paper. One significant update, however, also merits discussion. Both versions address software development process traceability, but DO-178C adds bidirectional traceability:

From DO-178B/C Chapter 5.5

DO-178B

Traceability

Traceability guidance includes:

a. Traceability between system requirements and software requirements should be provided to enable verification of the complete implementation of the system requirements and give visibility to the derived requirements.

b. Traceability between the low-level requirements and high-level requirements should be provided to give visibility to the derived requirements and the architectural design decisions made during the software design process, and allow verification of the complete implementation of the high-level requirements.

c. Traceability between Source Code and low-level requirements should be provided to enable verification of the absence of undocumented Source Code and verification of the complete implementation of the low-level requirements.

DO-178C

Software Development Process Traceability

Software development traceability activities include:

a. Trace Data, showing the bi-directional association between system requirements allocated to software and high-level requirements is developed. The purpose of this Trace Data is to:

    a. Enable verification of the complete implementation of the system requirements allocated to software.

    b. Give visibility to those derived high-level requirements that are not directly traceable to system requirements

b. Trace Data, showing the bi-directional association between the high-level requirements and low-level requirements is developed. The purpose of this Trace Data is to:

    a. Enable verification of the complete implementation of the high-level requirements allocated to software.

    b. Give visibility to those derived low-level requirements that are not directly traceable to high-level requirements

c. Trace Data, showing the bi-directional association between the low-level requirements and the source code is developed. The purpose of this Trace Data is to:

    a. Enable verification that no Source Code implements an undocumented function.

    b. Enable verification of the complete implementation of the low-level requirements.

For organizations that must be in compliance with DO-178C, the new requirement means that they will need a system that enforces policies and is flexible enough to provide bi-directional traceability. As of the writing of this paper, the TSOs required for FAA approval reference DO-178B, so vendors may not yet need to follow DO-178C. For more information about FAA approval, regulations, and standards, contact the FAA.

ENSURING NON-FUNCTIONAL REQUIREMENTS WITH POLICY DRIVEN DEVELOPMENT

In many manufacturing processes, there are known methods for ensuring that the end product not only functions as expected, but also exhibits strong quality characteristics. For example, a bicycle must not only have pedaling and braking functionality (functional requirements), but should also be able to withstand your body weight or meet standards for bicycle safety (non-functional requirements).

Bicycle manufacturers may automate the application of those practices in their factories, as well as automatically measure the efficacy of their production methods. Taken together, this would be an example of policy-driven manufacturing—the quality goals are clearly stated, tools and training are available to ensure that the goals can be met, and compliance with the goals is automatically enforced. The same principles should be applied to how software is engineered.

Developing embedded software, though, is arguably more complex than manufacturing a bicycle. This is because clearly policies are either lacking in many organizations or they don't have a way to monitor and enforce compliance with those policies. As a result, engineers are forced to make business-critical decisions every day. As software continues to play a greater role in systems responsible for functional safety, engineering decisions can lead to significant business risks.

DO-178B/C provides a detailed framework that can be used as a foundation for integrating a policy-driven software development strategy. Moreover, specifying the tasks that need to be accomplished in order to reduce risks forms the crux of the standard. The key to reining in these risks is to align software development activities with your organization's business goals. This can be achieved through "policy-driven development," which ensures that engineers deliver software according to your expectations.

Policy-driven development involves 1) clearly defining expectations and documenting them in understandable polices, 2) training the engineers on the business objectives driving those policies, and 3) enforcing the policies in an automated, unobtrusive way. By adopting a policy-driven strategy, businesses are able to accurate and objectively measure productivity and application quality, which lowers development costs and reduces risk.

With public safety, potential litigation, market position and other consequences on the line, it behooves software development teams and people in the traditional business management positions to come together on policy and implement the strategy into their software development lifecycle.

DEVELOPMENT TESTING FOR DO-178B/C

Manually driving policy throughout the development process isn't feasible in most organizations. Open source ALM tools are often coupled with code analysis tools and testing frameworks to create an ad-hoc development testing solution that delivers some insight into software engineering activities. But in a highly competitive market, being first to market with high-quality software engenders a significant advantage. Data from adhoc development testing infrastructures can deliver a significant volume of data, but rarely actionable analytics necessary to help development teams understand where to allocate resources to meet their goals efficiently.

Parasoft DTP for C and C++ applications is an integrated solution for automating software verification and validation processes and software quality tasks specified in DO-178B/C, including static analysis, data flow static analysis, metrics analysis, peer code review, unit testing and runtime error detection. This provides teams a practical way to prevent, expose, and correct errors in functional safety systems.

Parasoft prioritizes potential defects based on configurable severity assignments and automatically assigns them to the engineer responsible for the code. Direct links to the problematic code and a description of how to fix it is

distributed to his or her IDE. For embedded and cross-platform development, DTP for C and C++ can be used in both host-based and target-based code analysis and test flows.

- **Automated code analysis:** The policy-based static code analysis ensures that industry coding standards and development policies are being followed. Users can define custom rule sets or use the built-in rules to identify potential coding errors and policy violations.
- **Flow analysis:** Flow Analysis simulates possible execution paths and determines whether these paths could trigger specific categories of runtime errors. This is especially useful for systems in which detailed runtime analysis is ineffective or impossible, such as embedded systems.
- **Code Review:** DTP Change Explorer enables convenient analyses of source code deltas between specific milestones or points in development. Overlapping code delta information with static analysis or unit testing results elevates the traditional code review process to completely new level.
- **Runtime Error Detection:** The application under test is continuously monitored for problems such as memory leaks, null pointers, uninitialized memory, and buffer overflows.
- **Unit and integration testing with coverage analysis:** Automatically run unit tests generated by open frameworks and report results, including coverage data. You can also generate complete unit tests, including test drivers and test cases for individual functions, to validate functional behavior. Automatically generated test cases also check function responses to unexpected inputs. For a deeper discussion about execution environments, including simulated targets, read the following papers:
    - Techniques for Unit Testing Embedded Systems Software
    - Integrated Error-Detection Techniques: Find More Bugs in Embedded C Software
- **Configurable reports:** Generate HTML, PDF, and custom format reports that cover which files were tested and analyzed in addition to providing test results and code coverage. Automatically email reports using role-based filters, distribute data directly to developers responsible for the code, as well as send summary reports to managers and team leads.
- **Integration with Parasoft Development Testing Solution:** Parasoft DTP provides traceability of all project artifacts, requirements, defects/enhancements, and tasks so you can enforce policies related to compliance and process standardization. It identifies inefficiencies by tracking and analyzing software development metrics and progress, so development teams can build on the increased productivity.

DO-178B/C COMPLIANCE WITH PARASOFT

The following tables matches key principles from DO-178B/C with Parasoft DTP for C/C++. The table is intended to provide a sample of Parasoft functionality and is by no means complete. Visit our Software Development Standards and Compliance page for additional information on how Parasoft can help you achieve compliance with other industry standards.

5.3.2: Software Coding Process Activities

METHOD	PARASOFT APPROACH
The Source Code should implement the low-level requirements and conform to the software architecture. The Source Code should conform to the Software Code Standards.	Rules that enforce industry best practices are built in. Users can select entire libraries based on industry standards, individual rules, or create custom rules based on the organization's policies.

The Source Code should be traceable to the Design Description.

Inadequate or incorrect inputs detected during the software coding process should be provided to the software requirements process, software design.

Parasoft DTP, coupled with a suite of testing tools, such as Parasoft C/C++test, enables bi-directional traceability from code to requirement. This ensures that the source code is implemented correctly.

6.3.3: Reviews and Analyses of the Software Architecture

METHOD

The objective of these reviews and analyses is to detect and report errors that may have been introduced during the development of the software architecture. These reviews and analyses confirm that the software architecture satisfies these objectives:

- Compatibility with the high-level requirements: The objective is to ensure that the software architecture does not conflict with the high-level requirements, especially functions that ensure system integrity, for example, partitioning schemes.
- Consistency: The objective is to ensure that a correct relationship exists between the components of the software architecture. This relationship exists via data flow and control flow.
- Compatibility with the target computer: The objective is to ensure that no conflicts exist, especially initialization, asynchronous operation, synchronization and interrupts, between the software architecture and the hardware/ software features of the target computer.
- Verifiability: The objective is to ensure that the software architecture can be verified, for example, there are no unbounded recursive algorithms.
- Conformance to standards: The objective is to ensure that the Software Design Standards were followed during the software design process and that deviations to the standards are justified, especially complexity restrictions and design constructs that would not comply with the system safety objectives.
- Partitioning integrity: The objective is to ensure that partitioning breaches are prevented or isolated.

PARASOFT APPROACH

Configure Parasoft rules to enforce any kind of policy, including a policy that specifies software architecture and a policy based on target computer specifications.

Parasoft includes software development testing tools, such as static analysis, unit testing, code coverage analysis, peer review analysis, and runtime error detection to verify that the code performs as expected.

Rules that enforce industry best practices are built in. Users can select entire libraries based on industry standards, individual rules, or create custom rules based on the organization's policies.

6.3.4: Reviews and Analyses of the Source Code

METHOD

Source Code complies with low-level requirements.

Source Code complies with software architecture.

Source Code is verifiable Source Code conforms to standards.

Source Code traceable to low-level requirements.

Source code is accurate and consistent.

PARASOFT APPROACH

Parasoft can automatically generate test cases based on user definitions to ensure that code complies with stated low-level requirements.

Configure rules to enforce any kind of policy, including a policy based on software architecture. Parasoft includes software development testing tools, such as static analysis, unit testing, code coverage analysis, peer review analysis, and runtime error detection to verify that the code performs as expected.

Rules that enforce industry best practices are built into Parasoft Test. Users can select entire libraries based on industry standards, individual rules, or create custom rules based on the organization's policies.

Pattern-based data flow analysis checks all possible paths simulates test case execution and detects errors across multiple units, components, and files to ensure that the code is accurate and consistent.

Correlate requirements to tasks to code and other requirements to achieve full traceability.

6.4.1: Test Environment

METHOD

More than one test environment may be needed to satisfy the objectives for software testing.

An excellent test environment includes the software loaded into the target computer and tested in a high-fidelity simulation of the target computer environment.

Note: In many cases, the requirements-based coverage and structural coverage necessary can be achieved only with more precise control and monitoring of the test inputs and code execution than generally possible in a fully integrated environment. Such testing may need to be performed on a small software component that is functionally isolated from other software components.

Certification credit may be given for testing done using a target computer emulator or a host computer simulator. Guidance for the test environment includes:

- Selected tests should be performed in the integrated target computer environment, since some errors are only detected in this environment.

PARASOFT APPROACH

Parasoft has a number of test environment capabilities:

- It can be used with a wide variety of embedded RTOSes and architectures by cross-compiling generated test harness and code instrumentation for a desired target runtime environment.
- Full customization of the test execution sequence is also supported. In addition to using the built-in test automation, users can incorporate custom test scripts and shell commands to fit the tool into their specific build and test environment.

6.4.3: Requirements-based Testing Methods

METHOD

Requirements-Based Hardware/Software Integration Testing: Typical errors revealed by this testing method include:

- Incorrect interrupt handling.
- Failure to satisfy execution time requirements.
- Incorrect software response to hardware transients or hardware failures, for example, start-up sequencing, transient input loads and input power transients.
- Data bus and other resource contention problems, for example, memory mapping.
- Inability of built-in test to detect failures.
- Errors in hardware/software interfaces.
- Incorrect behavior of feedback loops.
- Incorrect control of memory management hardware or other hardware devices under software control.
- Stack overflow.
- Incorrect operation of mechanism(s) used to confirm the correctness and compatibility of field-loadable software.
- Violations of software partitioning.

Requirements-Based Software Integration Testing: Typical errors revealed by this testing method include:

- Incorrect initialization of variables and constants.
- Parameter passing errors.
- Data corruption, especially global data.
- Inadequate end-to-end numerical resolution.
- Incorrect sequencing of events and operations.

Requirements-Based Low-Level Testing: Typical errors revealed by this testing method include:

- Failure of an algorithm to satisfy a software requirement.

PARASOFT APPROACH

Parasoft's GUI helps you parameterize test cases and stubs—enabling increased test scope and coverage with minimal effort. Stub analysis and generation is facilitated by the Stub View, which presents all functions used in the code and allows users to create stubs for any functions not available in the test scope—or to alter existing functions for specific test purposes.

C/C++test can execute tests on the target, as well as in the test environment so that hardware integration can be verified. Software engineers can adjust the unit testing scope to include functions as they are completed and run regression tests to ensure proper software integration. Additionally, the runtime error detection capabilities expose integration defects and other errors as the application is exercised.

The advanced interprocedural static analysis module simulates feasible application execution paths and determines whether these paths could trigger specific categories of runtime bugs. Defects detected include:

- Using uninitialized or invalid memory
- Null pointer dereferencing
- Array and buffer overflows
- Division by zero
- Memory and resource leaks
- Dead code

Users can adjust static analysis rules in Parasoft C/C++test to analyze as much or as little of the code as needed. Files, as well as lines of code, can be tested based on when they were added, who added them, by path, etc. Software engineers can adjust the unit testing scope to include functions as they are completed and run regression tests to ensure proper software integration. Additionally, the runtime error detection capabilities expose integration defects and other errors as the application is exercised.

Low-level testing usually means testing single functions with given inputs described in the requirements. Users can parameterize and automatically generate unit test cases to functions as they are completed. Tests can be saved and ran as regression suites to ensure proper software integration.

- Incorrect loop operations.
- Incorrect logic decisions.
- Failure to process correctly legitimate combinations of input conditions.
- Incorrect responses to missing or corrupted input data.
- Incorrect handling of exceptions, such as arithmetic faults or violations of array limits.
- Incorrect computation sequence.
- Inadequate algorithm precision, accuracy or performance.

6.4.4.2: Structural Coverage Analysis

METHOD

Structural Coverage Analysis:

The objective of this analysis is to determine which code structure was not exercised by the requirements-based test procedures. The requirements-based test cases may not have completely exercised the code structure, so structural coverage analysis is performed and additional verification produced to provide structural coverage. Guidance includes:

- The analysis should confirm the degree of structural coverage appropriate to the software level.
- The structural coverage analysis may be performed on the Source Code unless the software level is A and the compiler generates object code that is not directly traceable to Source Code statements. Then, additional verification should be performed on the object code to establish the correctness of such generated code sequences. A compiler generated array-bound check in the object code is an example of object code that is not directly traceable to the Source Code.
- The analysis should confirm the data coupling and control coupling between the code components.

PARASOFT APPROACH

A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge the efficacy and completeness of the tests, as well as demonstrate compliance with test and validation requirements.

Back tracing from coverage elements to the corresponding test cases enables users to analyze test results and extend the test cases for better coverage, with higher efficiency.

All test results can be exported for compliance purposes.

Object Coverage module (aka Assembly Coverage) helps users to determine if there is any object code generated by the compiler which is not traceable to the source code and ensure that it was examined during testing procedures.

6.4.4.3: Structural Coverage Analysis Resolution

METHOD

Structural Coverage Analysis Resolution:

Structural coverage analysis may reveal code structure that was not exercised during testing. Resolution would require additional software verification process activity. This unexecuted code structure may be the result of:

- Shortcomings in requirements-based test cases or procedures: The test cases should be supplemented or test procedures changed to provide the missing coverage. The method(s) used to perform the requirements- based coverage analysis may need to be reviewed.
- Inadequacies in software requirements: The software requirements should be modified and additional test cases developed and test procedures executed.
- Dead code: The code should be removed and an analysis performed to assess the effect and the need for re-verification.
- Deactivated code: For deactivated code, which is not intended to be executed in any configuration used within an aircraft or engine, a combination of analysis and testing should show that the means by which such code could be inadvertently executed are prevented, isolated, or eliminated. For deactivated code, which is only executed in certain configurations of the target computer environment, the operational configuration needed for normal execution of this code should be established and additional test cases and test procedures developed to satisfy the required coverage objectives.

PARASOFT APPROACH

Parasoft can automatically analyze code and generate a unit-level test suite that achieves high coverage. Parasoft Test also provides technologies to help you increase the test coverage:

- Simple test case extensions.
- Flexible stubs framework makes tests more realistic.
- Data-driven testing with different sets of data (automatically generated or from a data source).

A Data Source Wizard helps parameterize test cases and stubs—enabling increased test scope and coverage with minimal effort. Stub analysis and generation is facilitated by the Stub View, which presents all functions used in the code and allows users to create stubs for any functions not available in the test scope—or to alter existing functions for specific test purposes.

The advanced interprocedural static analysis module simulates feasible application execution paths and determines whether these paths could trigger specific categories of runtime bugs.

Defects detected include:

- Using uninitialized or invalid memory
- Null pointer dereferencing
- Array and buffer overflows
- Division by zero
- Memory and resource leaks
- Dead code

12.2: Tool Qualification

METHOD

Qualification of a tool is needed when processes of this document are eliminated, reduced or automated by the use of a software tool without its output being verified as specified in section 6. The use of software tools to automate activities of the software life cycle processes can help satisfy system safety objectives insofar as they can enforce conformance with software development standards and use automatic checks.

Only deterministic tools may be qualified, that is, tools which produce the same output for the same input data when operating in the same environment. The tool qualification process may be applied either to a single tool or to a collection of tools.

The software verification process objectives for software development tools are described in paragraph 12.2.1 (see also RTCA document DO-330 for more information). A tool may be qualified only for use on a specific system where the intention to use the tool is stated in the Plan for Software Aspects of Certification. Use of the tool for other systems may need further qualification.

PARASOFT APPROACH

Qualifying Parasoft DTP quality tools involves running static analysis, flow analysis, unit tests, and any other testing activity used in your development process on the set of test cases with known and well-defined outcome provided by Parasoft. Parasoft will consistently, accurately and objectively report errors, which ensures that the tool functions properly.

Parasoft offers a qualification kit for C/C++test. Contact Parasoft Professional Services for additional information regarding the qualification kit.

SUMMARY

Developing DO-178B/C-compliant software for airborne systems is no easy feat. But Parasoft helps ease the burden by offering a broad range of development testing and verification tools, including coding standards compliance analysis, data and control flow analysis, unit testing, application monitoring, workflow components, and automated peer code review. When paired with a policy-driven development approach that bridges that gap between development activities and business processes, Parasoft DTP becomes a powerful tool for enforcing compliance with DO-178B/C and other standards.

Development teams can also generate configurable test reports that contain a high level of detail, which helps facilitate the traceability requirements for the software verification process. The testing functions, configurable contexts and reporting mechanisms in Parasoft DTP arm software development teams with proven tools that help them achieve DO-178B/C compliance.

www.parasoft.com

Parasoft Headquarters:
+1-626-256-3680

Parasoft EMEA:
+31-70-3922000

Parasoft APAC:
+65-6338-3628

PARASOFT®
Automated Software Testing