

Meeting DO-178B Software Verification Guidelines with Coverity Integrity Center

May, 2009

Executive Summary

Development organizations that create safety-critical airborne software systems must have the systems approved for use by the Federal Aviation Administration (FAA). The FAA recognizes DO-178B as an acceptable means of compliance for securing the Federal Aviation Administration's (FAA) approval of software in airborne systems and equipment. Created by RTCA, Inc., this standard provides guidelines for determining that the software aspects of airborne systems and equipment comply with airworthiness requirements. Many of the guidelines in the Software Verification and Software Lifecycle Data section of this standard can be supported with the use of the Coverity Integrity Center, a software analysis solution used to find critical defects throughout the software development life cycle. Coverity's software analysis works in conjunction with other testing methods (dynamic testing, requirements tracing, etc.) to enhance the software verification process by automating code review, significantly lowering cost of compliance. Additionally, by using automated software analysis as outlined below, development organizations can improve software integrity by eliminating defects early in the development lifecycle and improve productivity by shortening defect remediation and testing cycles.

This document will discuss how the Coverity Integrity Center can help with specific software verification guidelines as outlined in DO-178B.

Introduction

DO-178B is published by RTCA, a private, not-for-profit organization, and is intended for an international audience, so it uses generic terms and minimizes references to specific national regulations and procedures. It provides guidelines for the production of airborne systems equipment software. It also includes guidelines for determining if an airborne software system complies with specific system airworthiness requirements. These guidelines address the safety concerns of the aviation industry but aviation law does not mandate such guidelines. So what is DO-178B?

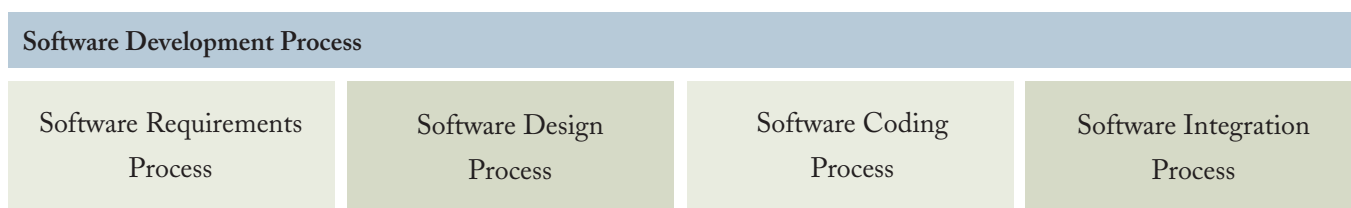
DO-178B is one of the most stringent standards in use in the software development industry. It offers a strict certification requirement for software where anomalous behavior could cause a catastrophic failure condition. Companies around the world use this standard to specify the safety and airworthiness of software for avionics systems. More specifically, it is used in the development, supply, acquisition, evaluation/certification, and operation of software products to be integrated in airborne systems and equipment. DO-178B focuses on the entire software lifecycle environment and describes techniques and methods appropriate to ensure the integrity and reliability of avionics software. The standard is considered so thorough that it has been often used outside the aeronautics industry and has been instrumental to the development of similar standards in other fields where dependability and safety is a strong requirement including: Nuclear Power, Rail and Automotive, and Medical industries.

What DO-178B is Not

Before discussing how software analysis pertains to specific sections of the standard, it is important to be aware of what the guideline is not about. DO-178B is a set of guidelines, not requirements. As mentioned above, it has been written in general terms so that it can be used internationally. Therefore, it does not mention country specific organizations, and does not refer to country specific regulations. Further, it does not provide a complete description of the system life cycle processes, including the system safety assessment and validation processes or aircraft and engine certification processes, nor does it cover operational aspects of software. DO-178B discusses certification issues only in relation to the software life cycle. This means that certification of software aspects such as user-modifiable data is beyond the scope of DO-178B. Lastly, it does not provide guidelines concerning the structure of the applicant organizations, personnel qualification criteria, the relationships between the applicant and its suppliers, or how the responsibilities are divided.

Focus on Software Lifecycle Process Verification

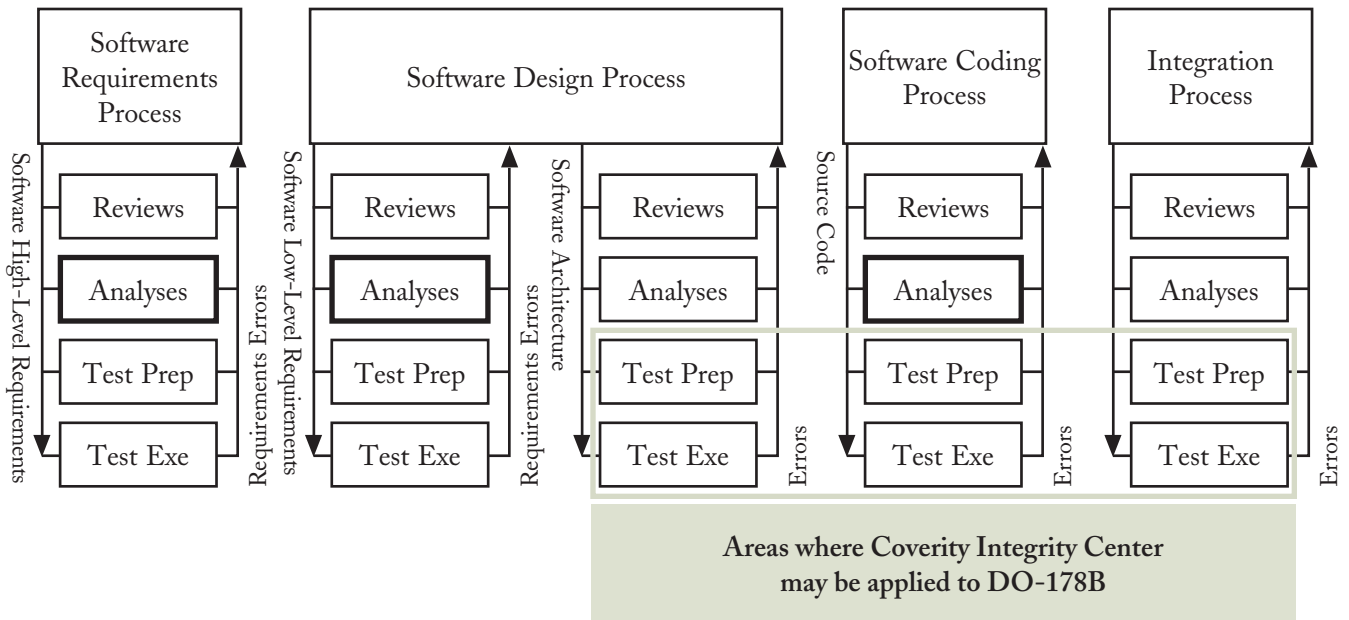
The DO-178B guideline focuses on verification to implement safety measures in the software lifecycle processes. Particular emphasis is given to system aspects relating to software development, defining the information flow between system and software life cycle processes, mainly for safety and verification matters. The guideline describes the Software Development Process in four phases:



The guideline also establishes a Software Verification Process Framework, consisting of three techniques to perform the validation of the Process:

- **Reviews:** Provide a qualitative assessment of correctness
- **Analyses:** Provide repeatable evidence of correctness
- **Tests:** Demonstrate that the software satisfies its high- and low-level requirements and provide a high degree of confidence that errors which could lead to unacceptable failure conditions have been removed.

The Verification Process is not performed at the end of the system delivery, but is integral in the entire process:
 The graphic below illustrates areas where Coverity Integrity Center may be applied to DO-178B.



Additionally, Table 1 below summarizes Coverity’s capabilities as they pertain to specific sections of DO-178B.

Table 1	
Coverity Integrity Center Capability	DO-178B Standard
Architecture Analysis	6.3.3, 6.3.4, 11.7
Static Analysis	6.3.3, 6.3.4, 6.4.3, 6.4.4.3, 11.7, 11.8
Build Analysis	7.2.5, 7.2.6

The remainder of this paper will discuss in more detail how the Coverity Integrity Center supports each of the sections listed above.

6.0 Software Verification Process

Software verification is used to determine if the software performs as intended. Section 6 of DO-178B “Software Verification” explains the steps necessary to verify airborne software systems, including the use of Architecture Analysis and Code Analysis. Coverity’s Architecture Analysis capability is used by developers and architects to automatically visualize architectural structure and dependencies in large, C/C++ and Java code bases. It provides deep insight into complexity and dependencies, allowing teams to understand the downstream impact of refactoring and to enforce design rules.

Coverity’ Static Analysis capability is used by development teams to find and eliminate quality, security, and concurrency defects in C/C++, C#, and Java code. It’s low false-positive rate and precision analysis techniques produce highly accurate results that make it the standard for static code analysis.

Coverity’s static and architecture analysis can be used to comply with the following areas of Section 6.0.

6.3.3 Reviews and Analyses of the Software Architecture.

This requires organizations to review their software architecture for potential errors introduced during its development, including satisfying numerous specific objectives. 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 the following objectives.

- a. 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. The reviewer can use Coverity’s Architecture Analysis to visualize and understand the dependencies and relationships among code components. With this information, reviewers can determine if the software architecture is compatible with the high level requirements.

- b. 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, which Coverity Architecture is able to analyze and depict.

c. 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. Coverity's Static Analysis capability can detect violations of this guideline.

The following checks can be used:

- PREVENT_UNINIT
- UNINIT_CTOR
- ATOMICITY
- LOCK, MISSING_LOCK
- ORDER_REVERSAL
- SLEEP

d. Verifiability: The objective is to ensure that the software architecture can be verified, for example, there are no unbounded recursive algorithms. Coverity Architecture Analysis can be used here.

e. Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, especially complexity restrictions and code constraints that would be consistent with the system safety objectives. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and the complexity of logical or numeric expressions. Coverity Architecture Analysis measures complexity and dependencies, allowing architects to understand areas that are overly complex or where unwanted dependencies exist. It also allows users to set design rules and flag violations of these rules, which would meet the guideline in the section.

f. Partitioning integrity: The objective is to ensure that partitioning breaches are prevented or isolated. Coverity Integrity Center Architecture Analysis provides for the creation and enforcement of rules establishing the partitioning of the software architecture, ensuring that partitions are coupled only to the ones set in the requirements, and flagging errors when they are not.

6.3.4 Reviews and Analyses of the Source Code

The objective of this section is to detect and report errors that may have been introduced during the software coding process. These reviews and analyses confirm that the outputs of the software coding process are accurate, complete and can be verified. Primary concerns include correctness of the code with respect to the software requirements and the software architecture, and conformance to the Software Code Standards. These reviews and analyses are usually confined to the Source Code.

b. Compliance with the software architecture: The objective is to ensure that the Source Code matches the data flow and control flow defined in the software architecture. As mentioned above, Coverity's Architecture Analysis can detect deviations from the desired architecture. This enables teams to ensure the code does not violate the design intent.

d. Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, especially complexity restrictions and code constraints that would be consistent with the system safety objectives. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and the complexity of logical or numeric expressions. This analysis also ensures that deviations to the standards are justified. The Coverity Integrity Center includes capabilities for meeting these objectives. Architecture Analysis can check for excess complexity, including fat and tangles, while Static Analysis can be extended to check for coding standards violations through the creation of custom checks.

f. Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, fixed point arithmetic overflow and resolution, resource contention, worse-case execution timing, exception handling, use of uninitialized variables or constants, unused variables or constants, and data corruption due to task or interrupt conflicts. Coverity's Static Analysis can detect these source code issues by performing the following checks:

- PREVENT STACK_USE
- OVERRUN_STATIC
- OVERRUN_DYNAMIC
- UNCAUGHT_EXCEPT
- UNINIT, UNINIT_CTOR
- UNUSED_VALUE
- PW.XXX (Compiler Warnings)

6.4.3 Requirements-Based Testing Methods

Requirements-based testing methods consist of methods for requirements-based hardware/software integration testing, requirements-based software integration testing, and requirements-based low-level testing. With the exception of hardware/software integration testing, these methods do not prescribe a specific test environment or strategy. This guidance includes:

b. Requirements-based Software Integration Testing: This testing method should concentrate on the inter-relationships between the software requirements, and on the implementation of requirements by the software architecture. The objective of requirements-based software integration testing is to ensure that the software components interact correctly with each other and satisfy the software requirements and software architecture. This method may be performed by expanding the scope of requirements through successive integration of code components with a corresponding expansion of the scope of the test cases. 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.

The following checks from Coverity's Static Analysis capability can meet this guideline:

- UNINIT
- UNINIT_CTOR
- PASS_BY_VALUE
- FORWARD_NULL
- REVERSE_INULL
- OVERRUN_STATIC
- OVERRUN_DYNAMIC
- USE_AFTER_FREE

c. Requirements-Based Low-Level Testing: This testing method should concentrate on demonstrating that each software component complies with its low-level requirements. The objective of requirements-based low-level testing is to ensure that the software components satisfy their low-level requirements. Typical errors revealed by this testing method include:

- Failure of an algorithm to satisfy a software requirement.
- 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.

The following checks from Coverity's Static Analysis capability can meet this guideline:

- INFINITE_LOOP
- NO_EFFECT
- INVALIDATE_ITERATOR
- MISMATCHED_ITERATOR
- DEADCODE, UNREACHABLE
- CHECKED_RETURN
- NULL_RETURNS
- UNUSED_VALUE
- MISSING_BREAK
- MISSING_RETURN

6.4.4.3 Structural Coverage Analysis Resolution

Structural coverage analysis may reveal code structure that was exercised during testing. Resolution would require additional software verification process activity. This unexecuted code structure may be the result of:

c. Dead code: The code should be removed and an analysis performed to assess the effect and the need for re-verification. Coverity Integrity Center can help meet this guideline. Architecture Analysis can be used to identify procedures or functions that are not dependent on any others pointing to possible dead code. Additionally, Static Analysis can be used to check for unused code. Specifically, it provides the following checks:

- DEADCODE
- UNREACHABLE
- callgraph-metrics.txt

d. 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. Architecture Analysis can be used to visualize the entire code structure. Static Analysis can be used to check for unused code. Specifically, it provides the following checks:

- DEADCODE
- UNREACHABLE
- callgraph-metrics.txt

7.2 Software Configuration Management Process Activities

The Software Configuration Management (SCM) process includes the activities of identification, change control, baseline establishment, and archiving of the software product, including the related software life cycle data. These guidelines define the objectives for each SCM process activity.

7.2.5 Change Review.

The objective of the change review activity is to ensure problems and changes are assessed, approved or disapproved, approved changes are implemented, and feedback is provided to affected processes through problem reporting and change control methods. The Build Analysis capability of the CIC can help with this guideline by providing a Bill of Materials for any assembled software system, which allows development teams to ensure that changes to a specific software system build have been made. Teams can use the BoM provided by Build Analyzer and compare it to that provided by the Source Code Management system to verify they are the same, and thus that specific changes have been made to the software code. Additionally, Build Analyzer can be used to provide an impact analysis of a proposed change as input into the Change Review.

7.2.6 Configuration Status Accounting.

The objective of this activity is to provide data for the configuration management of software life cycle processes with respect to configuration identification, baselines, problem reports, and change control. This includes change history, which can be provided by Coverity's build analysis capability. As mentioned in the previous section, Build Analyzer allows teams to verify changes have been made to a software system by creating a Bill of Materials, which can confirm the information provided by the change control system and/or the source code management system.

11.0 Software Lifecycle Data

Data that enables the software lifecycle processes, system or equipment certification, and post-certification modification of the software product must have specific characteristics, form, configuration management controls, and content. This section describes these guidelines.

11.7 Software Design Standards.

The purpose of Software Design Standards is to define the methods, rules and tools to be used to develop the software architecture and low-level requirements. These standards should include:

- b. Naming conventions to be used. Coverity's Static Analysis can be extended to create custom checks for detecting violations of naming conventions used in code.

e. Constraints on design, for example, exclusion of recursion, dynamic objects, data aliases, and compact expressions. Coverity's Static Analysis can be extended to create custom checks for detecting violations of these constraints.

f. Complexity restrictions, for example, maximum level of nested calls or conditional structures, use of unconditional branches, and number of entry/exit points of code components. The Coverity Integrity Center can be used for measuring complexity. Architecture Analysis provides visualization and data on structural complexity. It can then be used to enforce restrictions on complexity by flagging violations. The Static Analysis capability provides metrics for cyclomatic complexity and Halstead complexity.

11.8 Software Code Standards.

The purpose of the Software Code Standards is to define the programming language, methods, rules and tools to be used to code the software. These standards should include:

a. Programming language(s) to be used and/or defined subset(s). For a programming language, reference the data that unambiguously defines the syntax, the control behavior, the data behavior and side-effects of the language. This may require limiting the use of some features of a language. Coverity's Static Analysis can be extended to create custom checks for detecting violations of this requirement.

c. Naming conventions for components, subprograms, variables, and constants. Coverity's Static Analysis can be extended to create custom checks for detecting violations of this requirement.

d. 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. Coverity's Static Analysis can be extended to create custom checks for detecting violations of this requirement.

Summary

Organizations that develop safety critical software for use in airborne systems, or any other safety-critical device or application, will benefit from using solutions that help automate compliance with DO-178B. Coverity Integrity Center is used by companies around the world to find and eliminate defects throughout the software development life cycle. Specifically, with regard to DO-178B compliance, companies can use Coverity's Architecture Analysis, Static Analysis and Build Analysis capabilities to measure and ensure the integrity of software systems by enforcing architectural design and intent, and identifying and eliminating defects related to coding standards as well as those that can result in system slowdowns or failures. Finally, teams can utilize Coverity's Build Analysis to verify change and ensure assembled code meets integrity standards.

Free Trial

Request a free Coverity trial and see first-hand how to rapidly detect and remediate serious defects and vulnerabilities. No changes to your code are necessary. There are no limitations on code size, and you will receive a complimentary report detailing actionable analysis results. Register for the on-site evaluation at www.coverity.com or call us at (800) 873-8193.

About Coverity

Coverity (www.coverity.com), the software integrity company, is the trusted standard for companies that have a zero tolerance policy for software failures, problems, and security breaches. Coverity's award winning portfolio of software integrity products helps customers prevent software problems throughout the application lifecycle. Over 100,000 developers and 600 companies rely on Coverity to help them ensure the delivery of high integrity. Coverity is a privately held company headquartered in San Francisco with offices in 6 countries and more than 130 employees.

Coverity and Coverity Prevent are trademarks of Coverity, Inc. All other company and product names are the property of their respective owners.

Headquarters

185 Berry Street, Suite 2400
San Francisco, CA 94107
(800) 873-8193
<http://www.coverity.com>
sales@coverity.com

Boston

230 Congress Street
Suite 303
Boston, MA 02110
(617) 933-6500

UK

Coverity Limited
Magdalen Centre
Robert Robinson Avenue
The Oxford Science Park
Oxford OX4 4GA
England

Japan

Coverity Asia Pacific
Level 32, Shinjuku Nomura Bldg.
1-26-2 Nishi-Shinjuku, Shinjuku-ku
Tokyo 163-0532
Japan