RTCA /DO-178

Table of Contents

1 RTCA/DO- 178

1.1 Introduction

The full form of RTCA/DO-178 is "Radio Technical Commission for Aeronautics/Document DO-178." It is a set of guidelines and standards for the certification of software used in airborne systems, particularly in the context of avionics. These standards are often followed by aviation authorities, such as the Federal Aviation Administration (FAA) in the United States, to ensure the safety and reliability of software in aircraft systems.

Here are the key types or versions of RTCA/DO-178:

1. **RTCA/DO-178 (1982):** The original version often referred to as DO-178A, provided guidelines for software considerations in airborne systems. It set the foundation for subsequent versions.
2. **RTCA/DO-178B (1992):** This is one of the most widely known and used versions of the standard. It expanded and improved upon the original DO-178A and provided more detailed guidance on software development and verification in aviation.
3. **RTCA/DO-178C (2011):** DO-178C is the most recent and current version of the standard. It supersedes DO-178B and incorporates modern software engineering practices and technologies. DO-178C places a stronger emphasis on software modeling, formal methods, and object-oriented technologies. It also aligns with other related standards like DO-254 for hardware considerations and DO-278 for integrated modular avionics.

While these are the primary versions, there may be amendments or supplements that address specific issues or technologies. It's essential for organizations in the aerospace industry to stay up to date with the latest version and any relevant supplements or amendments to ensure compliance with current industry standards and regulations.

1.2 Differences between DO-178B and DO-178A

This is a complete rewrite of DO-178A.

It is suggested that the entire document be read before using any of the sections or tables in isolation.

DO-178B is primarily a process-oriented document. For each process, objectives are defined and a means of satisfying these objectives are described. A description of the software life cycle data which shows that the objectives have been satisfied is provided. The objectives for each process are summarized in tables and the effect of software level on the applicability and independence of these objectives is specified in the tables. The variation by software level in configuration management rigor for the software life cycle data is also in the tables.

DO-178B recognizes that many different software life cycles are acceptable for developing software for airborne systems and equipment. DO-178B emphasizes that the chosen software life cycle(s) should be defined during the planning for a project. The processes which comprise a software development project, no matter which software life cycle was chosen, are described. These processes fall into three categories: the software planning process, the software development processes, which include software requirements, software design, software coding and integration; and the integral processes which include software verification, software quality assurance, and software configuration management and certification liaison. Integral processes are active throughout the software life cycle.

DO-178B requires that criteria which control the transitions between software life cycle processes should be established during the software planning process.

The relationship between the system life cycle processes and software life cycle processes is further defined. Failure conditions associated with functions which are to be implemented in software need to be considered during the system safety assessment process. This is the basis for establishing the software level. The software levels are now Level A, Level B, Level C, Level D, and Level E.

The software verification section emphasizes requirements-based testing, which is supplemented with structural coverage.

The items that are to be delivered and other data items are further defined, as well as the configuration management required

2 RTCA/DO- 178B

2.1 Introduction

The rapid increase in the use of software in airborne systems and equipment used on aircraft and engines in the early 1980s resulted in a need for industry-accepted guidance for satisfying airworthiness requirements. DO-178, "Software Considerations in Airborne Systems and Equipment Certification," was written to satisfy this need.

2.2 Purpose

The purpose of this document is to provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements. These guidelines are in the form of:

- Objectives for software life cycle processes.
- Descriptions of activities and design considerations for achieving those objectives ·
- Descriptions of the evidence that indicate that the objectives have been satisfied.

2.3 Scope

This document discusses those aspects of airworthiness certification that pertain to the production of software for airborne systems and equipment used on aircraft or engines. In discussing those aspects, the system life cycle and its relationship with the software life cycle is described to aid in the understanding of the certification process. A complete description of the system life cycle processes, including the system safety assessment and validation processes, or aircraft and engine certification process is not intended.

2.4 System Aspects Relating To Software Development

This section discusses those aspects of the system life cycle processes necessary to understand the software life cycle processes. Discussed are:

- Exchange of data between the system and software life cycle processes.
- Categorization of failure conditions, definition of software levels, and software level determination.

- System architectural considerations.
- System considerations for user-modifiable software, option-selectable software, and commercial off the-shelf software.
- System design considerations for field-loadable software.
- System requirements considerations for software verification.
- Software considerations in system verification.

2.4.1 Information Flow between System and Software Life Cycle Processes

The safety aspects of the information flow between system life cycle processes and the software life cycle processes. Due to interdependence of the system safety assessment process and the system design process, the flow of information described in these sections is iterative.

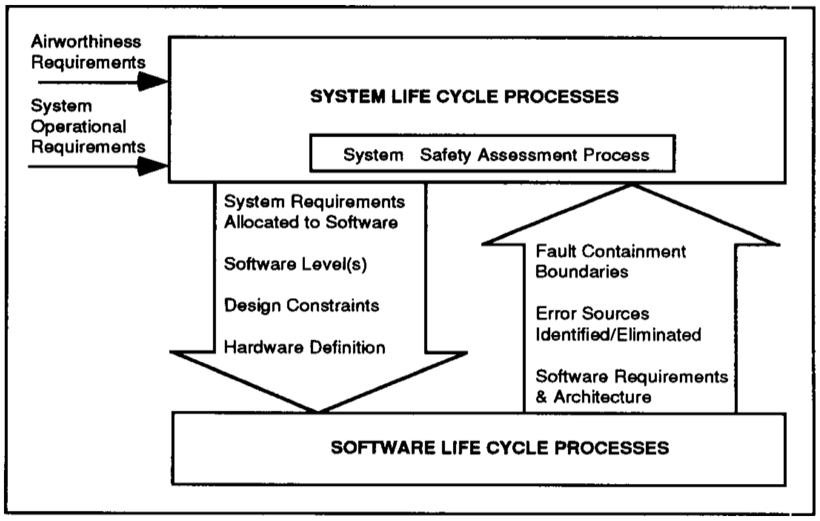
Airworthiness Requirements
System Operational Requirements
SYSTEM LIFE CYCLE PROCESSES
System Safety Assessment Process
System Requirements Allocated to Software
Software Level(s)
Design Constraints
Hardware Definition
Fault Containment Boundaries
Error Sources Identified/Eliminated
Software Requirements & Architecture
SOFTWARE LIFE CYCLE PROCESSES

Figure 1: System Safety-Related Information Flow between System and Software Life Cycle Processes

2.5 Software Level Definitions

Software level is based upon the contribution of software to potential failure conditions as determined by the system safety assessment process. The software level implies that the level of effort required showing compliance with certification requirements varies with the failure condition category. The software level definitions are:

- Level A: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
- Level B: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition for the aircraft.
- Level C: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.
- Level D: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
- Level E: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.

2.6 Software Level Determination

Initially, the system safety assessment process determines the software level(s) appropriate to the software components of a particular system without regard to system design. The impact of failure, both loss of function and malfunction, is addressed when making this determination.

2.7 System Architectural Considerations

If the system safety assessment process determines that the system architecture precludes anomalous behavior of the software from contributing to the most severe failure condition of a system, then the software level is determined by the most severe category of the remaining failure conditions to which the anomalous behavior of the software can contribute. The system safety assessment process considers the architectural design decisions to determine whether they affect software level or software functionality. Guidance is provided on several architectural strategies that may limit the impact of errors, or detect errors and provide acceptable system responses to contain the errors.

2.8 System Requirements Considerations for Software Verification

The system requirements are developed from the system operational requirements and the safety-related requirements that result from the system safety assessment process. Considerations include:

a. The system requirements for airborne software establish two characteristics of the software:
   - The software performs specified functions as defined by the system requirements.
   - The software does not exhibit specific anomalous behavior(s) as determined by the system safety assessment process. Additional system requirements are generated to eliminate the anomalous behavior.

b. These system requirements should then be developed into software high-level requirements that are verified by the software verification process activities.

2.9 Software Considerations in System Verification

Guidance for system verification is beyond the scope of this document. However, the software life cycle processes aid and interact with the system verification process. Software design details that relate to the system functionality need to be made available to aid system verification.

System verification may provide significant coverage of code structure. Coverage analysis of system verification tests may be used to achieve the coverage objectives of various test activities described under software verification.

2.10 Software Life Cycle

Software Life Cycle is determined by attributes of the project, such as system functionality and complexity, software size and complexity, requirements stability, use of previously developed results, development strategies and hardware availability. The usual sequence through the software development processes is requirements, design, coding and integration.

The processes of a software life cycle may be iterative, that is, entered and re-entered. The timing and degree of iteration varies due to the incremental development of system functions, complexity, requirements development, hardware availability, feedback to previous processes, and other attributes of the project.

2.10.1 Software Life Cycle Processes

The software life cycle processes are:

- The software planning process that defines and coordinates the activities of the software development and integral processes for a project.
- The software development processes that produce the software product. These processes are the software requirements process, the software design process, the software coding process, and the integration process.
- The integral processes that ensure the correctness, control, and confidence of the software life cycle processes and their outputs. The integral processes are the software verification process, the software configuration management process, the software quality assurance process, and the certification liaison process. It is important to understand that the integral processes are performed Concurrently with the software development processes throughout the software life cycle.

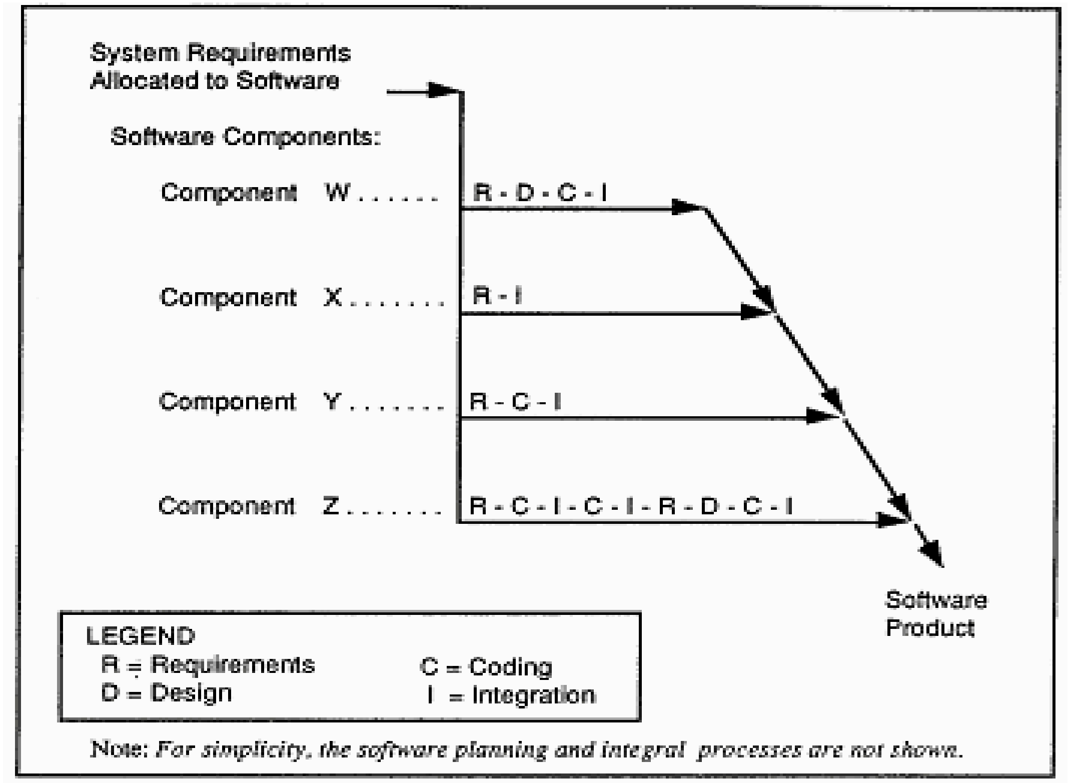
System Requirements
Allocated to Software
Software Components:
Component W . . . . . .
R - D - C - I
Component X . . . .
R - I
Component Y . . . .
R - C - I
Component Z . . . .
R - C - I - C - I - R - D - C - I
Software
Product
LEGEND
R = Requirements
D = Design
C = Coding
I = Integration
Note: For simplicity, the software planning and integral processes are not shown.

Figure 2: Example of Software Projects using four different development sequences

2.11 Software Planning Process

The objectives and activities of the software planning process produces the software plans and standards that direct the software development processes and the integral processes.

2.11.1 Software Planning Process Objectives

The purpose of the software planning process is to define the means of producing software which will satisfy the system requirements and provide the level of confidence which is consistent with airworthiness requirements. The objectives of the software planning process:

a. The activities of the software development processes and integral processes of the software life cycle that will address the system requirements and software level(s) are defined.

b. The software life cycle(s), including the inter-relationships between the processes, theirsequencing, feedback mechanisms, and transition criteria are determined.
c. The software life cycle environment, including the methods and tools to be used for theactivities of each software life cycle process has been selected.
d. Software development standards consistent with the system safety objectives for the software tobe produced are defined.

2.11.2 Software Plans

The purpose of the software plans is to define the means of satisfying the objectives of this document. They specify the organizations that will perform those activities. The software plans are:

- The Plan for Software Aspects of Certification serves as the primary means for communicating the proposed development methods to the certification authority for agreement, and defines the means of compliance with this document.
- The Software Development Plan defines the software life cycle (s) and software development environment.
- The Software Verification Plan defines the means by which the software verification process objectives will be satisfied.
- The Software Configuration Management Plan defines the means by which the software configuration management process objectives will be satisfied.

- The Software Quality Assurance Plan defines the means by which the software quality assurance process objectives will be satisfied.

Guidance for the software plans includes:

a. The software plans should comply with this document.

b. The software plans should define the criteria for transition between software life cycle processes by specifying:

    1. The inputs to the process, including feedback from other processes.
    2. Any integral process activities that may be required to act on these inputs.
    3. Availability of tools, methods, plans and procedures.

c. The software plans should state the procedures to be used to implement software changes prior to use on a certified aircraft or engine. Such changes may be as a result of feedback from other processes and may cause a change to the software plans.

2.12 Software Life cycle Environment Planning

The purpose of the planning for the software life cycle environment is to define the methods, tools, procedures, programming languages and hardware that will be used to develop, verify, control and produce the software life cycle data and software product.

- The methods and notations used in the software requirements process and software design process.
- The programming language(s) and methods used in the software coding process.
- The software development environment tools.
- The software verification and software configuration management tools.
- The need for tool qualification.

2.13 Software Development Environment

The software development environment is a significant factor in the production of high quality software. The software development environment can also adversely affect the production of airborne software in several ways. For example, a compiler could introduce errors by producing a corrupted output or a linker could fail to reveal a memory allocation error that is present. Guidance for the selection of software development environment methods and tools includes:

a. During the software planning process, the software development environment should be chosen to minimize its potential risk to the final airborne software.
b. The use of qualified tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together.
c. Software verification process activities or software development standards, which include consideration of the software level, should be defined to minimize potential software development environment-related errors.
d. If certification credit is sought for use of the tools in combination, the sequence of operation of the tools should be specified in the appropriate plan.
e. If optional features of software development tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan.

2.14 Software Test Environment

The purpose of software test environment planning is to define the methods, tools, procedures and hardware that will be used to test the outputs of the integration process. Testing may be performed using the target computer, a target computer emulator or a host computer simulator. Guidance includes:

a. The emulator or simulator may need to be qualified.
b. The differences between the target computer and the emulator or simulator, and the effects of these differences on the ability to detect errors and verify functionality, should be considered. Detection of those errors should be provided by other software verification process activities and specified in the Software Verification Plan.
c. The software development standards should comply.
d. The software development standards should enable software components of a given software product or related set of products to be uniformly designed and implemented.
e. The software development standards should disallow the use of constructs or methods that produce outputs that cannot be verified or that are not compatible with safety-

related requirements.

2.15 Software Development Process

This section discusses the objectives and activities of the software development processes. The software development processes are applied as defined by the software planning process and the Software Development Plan. The software development processes are:

- Software requirements process.
- Software design process.
- Software design process.
- Integration process.

Software development processes produce one or more levels of software requirements. High-level requirements are produced directly through analysis of system requirements and system architecture. Usually, these high-level requirements are further developed during the software design process, thus producing one or more successive, lower levels of requirements. However, if Source Code is generated directly from high-level requirements, then the high-level requirements are also considered low-level requirements and the guidelines for low-level requirements also apply.

The development of a software architecture involves decisions made about the structure of the software. During the software design process, the software architecture is defined and low-level requirements are developed. Low-level requirements are software requirements from which Source Code can be directly implemented without further information.

Each software development process may produce derived requirements. Derived requirements are requirements that are not directly traceable to higher level requirements. An example of such a derived requirement is the need for interrupt handling software to be developed for the chosen target computer. High-level requirements may include derived requirements, and low-level requirements may include derived requirements. The effects of derived requirements on safety related requirements are determined by the system safety assessment process.

2.15.1 Software Requirements Process

The software requirements process uses the outputs of the system life cycle process to develop the software high-level requirements. These high-level requirements include functional, performance, interface and safety-related requirements.

2.15.1.1 Software Requirements Process Objectives

The objectives of the software requirements process are:

a. High-level requirements are developed.
b. Derived high-level requirements are indicated to the system safety assessment process.

2.15.1.1.1 Software Requirements Process Activities

Inputs to the software requirements process include the system requirements, the hardware interface and system architecture (if not included in the requirements) from the system life cycle process, and the Software Development Plan and the Software Requirements Standards from the software planning process. When the planned transition criteria have been satisfied, these inputs are used to develop the software high-level requirements.

The primary output of this process is the Software Requirements Data

The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Guidance for this process includes:

a. The system functional and interface requirements that are allocated to software should be analyze for ambiguities, inconsistencies and undefined conditions.
b. Inputs to the software requirements process detected as inadequate or incorrect should be reported as feedback to the input source processes for clarification or correction.
c. Each system requirement that is allocated to software should be specified in the high-level requirements.
d. High-level requirements that address system requirements allocated to software to preclude system hazards should be defined.
e. The high-level requirements should conform to the Software Requirements Standards, and be verifiable and consistent.
f. The high-level requirements should be stated in quantitative terms with

tolerances where applicable.

g. The high-level requirements should not describe design or verification detail except for specified and justified design constraints.
h. Each system requirement allocated to software should be traceable to one or more software high- level requirements.
i. Each high-level requirement should be traceable to one or more system requirements, except for derived requirements.
j. Derived high-level requirements should be provided to the system safety assessment process.

2.15.2 Software Design Process

The software high-level requirements are refined through one or more iterations in the software design process to develop the software architecture and the low-level requirements that can be used to implement Source Code.

2.15.2.1 Software Design Process Objectives

The objectives of the software design process are:

a. The software architecture and low-level requirements are developed from the high-level requirements.
b. Derived low-level requirements are provided to the system safety assessment process.

2.15.2.1.1 Software Design Process Activities

The software design process inputs are the Software Requirements Data, the Software Development Plan and the Software Design Standards. When the planned transition criteria have been satisfied, the high- level requirements are used in the design process to develop software architecture and low-level requirements. This may involve one or more lower levels of requirements.

The primary output of the process is the Design Description which includes the software architecture and the low-level requirements.

The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Guidance for this process includes:

a. Low-level requirements and software architecture developed during the software design process should conform to the Software Design Standards and be traceable, verifiable and consistent
b. Derived requirements should be defined and analyzed to ensure that the higher level requirements are not compromised.
c. Software design process activities could introduce possible modes of failure into the software or, conversely, preclude others. The use of partitioning or other architectural means in the software design may alter the software level assignment for some components of the software.
d. In such cases, additional data should be defined as derived requirements and provided to the system safety assessment process.
e. Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks and cross-channel comparisons.
f. Responses to failure conditions should be consistent with the safety-related requirements.
g. Inadequate or incorrect inputs detected during the software design process should be provided to either the system life cycle process, the software requirements process, or the software planning process as feedback for clarification or correction.

2.15.3 Software Coding Process

In the software coding process, the Source Code is implemented from the software architecture and the low-level requirements.

2.15.3.1 Software Coding Process Objectives

The objective of the software coding process is:

a. Source code is developed that is traceable, verifiable, consistent, and correctly implements low- level requirements.

2.15.3.1.1 Software Coding Process Activities

The coding process inputs are the low-level requirements and software architecture from the software design process, and the Software Development Plan and the Software Code Standards. The software coding process may be entered or re-entered when the planned transition criteria are satisfied. The Source Code is produced by this process based upon the software architecture and the low-level requirements. The primary results of this process are Source Code and object code. The software coding process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Guidance for this process includes:

a. The Source Code should implement the low-level requirements and conform to the software architecture.
b. The Source Code should conform to the Software Code Standards.
c. The Source Code should be traceable to the Design Description.
d. Inadequate or incorrect inputs detected during the software coding process should be provided to the software requirements process, software design process or software planning process as feedback for clarification or correction.

2.15.4 Integration Process

The target computer, and the Source Code and object code from the software coding process are used with the linking and loading data in the integration process to develop the integrated airborne system or equipment.

2.15.4.1 Integration Process Objectives

The objective of the integration process is:

a. The Executable Object Code is loaded into the target hardware for hardware/software integration.

2.15.4.1.1 Integration Process Activities

The integration process consists of software integration and hardware/software integration. The integration process may be entered or re-entered when the planned transition criteria have been satisfied. The integration process inputs are the software architecture from the software design process, and the Source Code and object code from the software coding process.

The outputs of the integration process are the Executable Object Code and the linking and loading data. The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Guidance for this process includes:

a. The Executable Object Code should be generated from the Source and linking and loading data.
b. The software should be loaded into the target computer for hardware/software integration.
c. Inadequate or incorrect inputs detected during the integration process should be provided to the software requirements process, the software design process, the software coding process or the software planning process as feedback for clarification or correction.

2.15.5 Traceability

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.

2.15.6 Software Verification Process

The objectives and activities of the software verification process. Verification is a technical assessment of the results of both the software development processes and the software verification process. The software verification process is applied as defined by the software planning process and the Software Verification Plan .

Verification is not simply testing. Testing, in general, cannot show the absence of errors. As a result, the following subsections use the term "verify" instead of "test" when the software verification process objectives being discussed are typically a combination of reviews, analyses and test.

2.15.7 Software Verification Process Objectives

The purpose of the software verification process is to detect and report errors that may have been introduced during the software development processes. Removal of the errors is an activity of the software development processes. The general objectives of the software verification process are to verify that:

a. The system requirements allocated to software have been developed into software high-level requirements that satisfy those system requirements.
b. The high-level requirements have been developed into software architecture and low-level
c. Requirements that satisfy the high-level requirements. If one or more levels of software requirements are developed between high-level requirements and low-level requirements, the successive levels of requirements are developed such that each successively lower level satisfies its higher level requirements. If code is generated directly from high-level requirements, this objective does not apply.
d. The software architecture and low-level requirements have been developed into SourceCode that satisfies the low-level requirements and software architecture.
e. The Executable Object Code satisfies the software requirements.
f. The means used to satisfy these objectives are technically correct and complete for the software level.

2.15.8 Software Verification Process Activities

Software verification process objectives are satisfied through a combination of reviews, analyses, the development of test cases and procedures, and the subsequent execution of those test procedures. Reviews and analyses provide an assessment of the accuracy, completeness, and verifiability of the software requirements, software architecture, and Source Code. The development of test cases may provide further assessment of the internal consistency and completeness of the requirements. The execution of the test procedures provides a demonstration of compliance with the requirements.

The inputs to the software verification process include the system requirements, the software requirements and architecture, traceability data, Source Code, Executable Object Code, and the Software Verification Plan.

The outputs of the software verification process are recorded in Software Verification Cases and Procedures and Software Verification Results .

The need for the requirements to be verifiable once they have been implemented in the software may itself impose additional requirements or constraints on the software development processes.

The verification process provides traceability between the implementation of the software requirements and verification of those software requirements:

- The traceability between the software requirements and the test cases is accomplished by the requirements -based coverage analysis.
- The traceability between the code structure and the test cases is accomplished by the structural coverage analysis.

Guidance for the software verification activities includes:

a. High-level requirements and traceability to those high-level requirements should be verified.

b. The results of the traceability analyses and requirements-based and structural coverage analyses should show that each software requirement is traceable to the code that implements it and to the review, analysis, or test case that verifies it.

c. If the code tested is not identical to the airborne software, those differences should be specified and justified.

d. When it is not possible to verify specific software requirements by exercising the software in a realistic test environment, other means should be provided and their justification for satisfying the software verification process objectives defined in the Software Verification Plan or Software Verification Results.

e. Deficiencies and errors discovered during the software verification process should be reported to the software development processes for clarification and correction.

2.15.9 Software Reviews and Analyses

Reviews and analyses are applied to the results of the software development processes and software verification process. One distinction between reviews and analyses is that analyses provide repeatable evidence of correctness and reviews provide a qualitative assessment of correctness. A review may consist of an inspection of an output of a process guided by a checklist or similar aid. An analysis may examine in detail the functionality, performance, traceability and safety implications of a software component, and its relationship to other components within the airborne system or equipment.

2.15.10 Reviews and Analyses of the High-Level Requirements

The objective of these reviews and analyses is to detect and report requirements errors that may have been introduced during the software requirements process. These reviews and analyses confirm that the high-level requirements satisfy these objectives:

a. **Compliance with system requirements:** The objective is to ensure that the system functions to be performed by the software are defined, that the functional, performance, and safety- related requirements of the system are satisfied by the software high-level requirements, and that derived requirements and the reason for their existence are correctly defined.

b. **Accuracy and consistency:** The objective is to ensure that each high-level requirement is accurate, unambiguous and sufficiently detailed and that the requirements do not conflict with each other.

c. **Compatibility with the target computer:** The objective is to ensure that no conflicts exist between the high-level requirements and the hardware/software features of the target computer, especially, system response times and input/output hardware.

d. **Verifiability:** The objective is to ensure that each high-level requirement can be verified.

e. **Conformance to standards:** The objective is to ensure that the Software Requirements Standards were followed during the software requirements process and that deviations from the standards are justified.

f. **Traceability:** The objective is to ensure that the functional, performance, and safety-related requirements of the system that are allocated to software were developed into the software high-level requirements.

g. **Algorithm aspects:** The objective is to ensure the accuracy and behavior of the proposed algorithms, especially in the area of discontinuities

2.15.11 Reviews and Analyses of the Low-Level Requirements

The objective of these reviews and analyses is to detect and report requirements errors that may have been introduced during the software design process. These reviews and analyses confirm that the software low-level requirements satisfy these objectives:

a. **Compliance with high-level requirements:** The objective is to ensure that the software low- level requirements satisfy the software high-level requirements and that derived requirements and the design basis for their existence are correctly defined.

b. **Accuracy and consistency:** The objective is to ensure that each low-level requirement is accurate and unambiguous and that the low-level requirements do not conflict with each other.

c. **Compatibility with the target computer:** The objective is to ensure that no conflicts exist between the software requirements and the hardware/software features of the target computer, especially, the use of resources (such as bus loading), system response times, and input/output hardware.

d. **Verifiability:** The objective is to ensure that each low-level requirement can be verified.

e. **Conformance to standards:** The objective is to ensure that the Software Design Standards were followed during the software design process, and that deviations from the standards are justified.

f. **Traceability:** The objective is to ensure that the high-level requirements and derived requirements were developed into the low-level requirements.

g. **Algorithm aspects:** The objective is to ensure the accuracy and behavior of the proposed algorithms, especially in the area of discontinuities.

2.15.12 Reviews and Analyses of the Software Architecture

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:

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.

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.

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.

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

e. **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.

f. **Partitioning integrity:** The objective is to ensure that partitioning breaches are prevented or isolated.

2.15.13 Reviews and Analyses of the Source Code

The objective 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. The topics include:

a. **Compliance with the low-level requirements:** The objective is to ensure that the Source Code is accurate and complete with respect to the software low-level requirements, and that no Source Code implements an undocumented function.

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.

c. **Verifiability**: The objective is to ensure the Source Code does not contain statements and structures that cannot be verified and that the code does not have to be altered to test it.

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.

e. **Traceability:** The objective is to ensure that the software low-level requirements were developed into Source Code.

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, worst-case execution timing, exception handling, use of uninitialized variables or constants, unused variables or constants, and data corruption due to task or interrupt conflicts.

2.16 Software Testing Process

Testing of airborne software has two complementary objectives. One objective is to demonstrate that the software satisfies its requirements. The second objective is to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed.

- **Hardware/software integration testing:** To verify correct operation of the software in the target computer environment.
- **Software integration testing:** To verify the interrelationships between software requirements and components and to verify the implementation of the software requirements and software components within the software architecture.
- **Low-level testing:** To verify the implementation of software low-level requirements.

To satisfy the software testing objectives:

a. Test cases should be based primarily on the software requirements.
b. Test cases should be developed to verify correct functionality and to establish conditions that reveal potential errors.
c. Software requirements coverage analysis should determine what software requirements were not tested.
d. Structural coverage analysis should determine what software structures were not exercised.

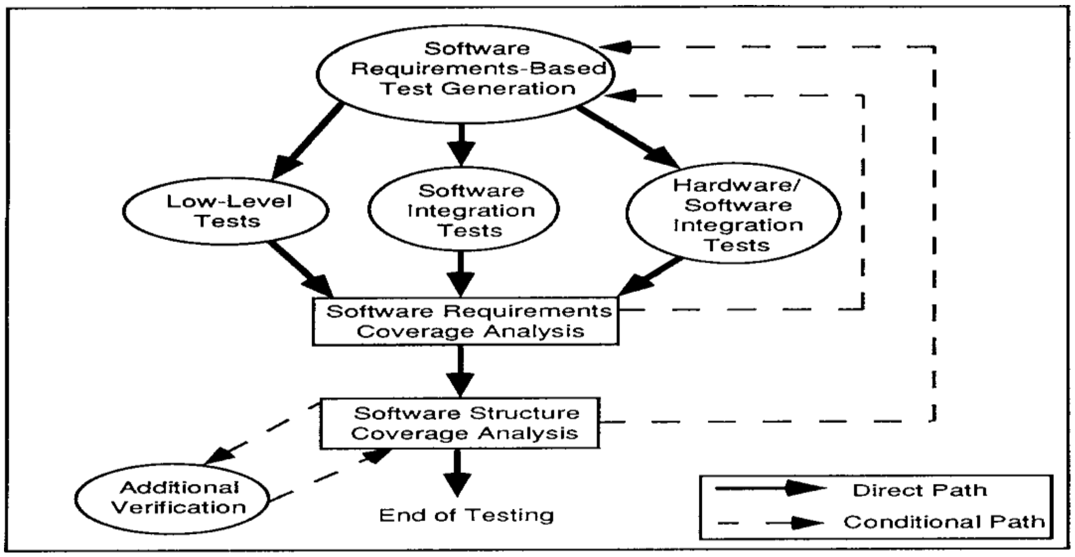
Software Requirements-Based Test Generation
Low-Level Tests
Software Integration Tests
Hardware/ Software Integration Tests
Software Requirements Coverage Analysis
Software Structure Coverage Analysis
Additional Verification
End of Testing
Direct Path
Conditional Path

Figure 3: Software testing process

2.17 Software Configuration Management Process

The objectives and activities of the software configuration management (SCM process). The SCM process is applied as defined by the software planning process (section 4) and the Software Configuration Management Plan Outputs of the SCM process are recorded in Software Configuration Management Records or in other software life cycle data.

2.18 Software Configuration Management Process Objectives

The SCM process, working in cooperation with the other software life cycle processes, assists in satisfying general objectives to:

a. Provide a defined and controlled configuration of the software throughout the software life cycle.
b. Provide the ability to consistently replicate the Executable Object Code for software manufacture or to regenerate it in case of a need for investigation or modification.
c. Provide control of process inputs and outputs during the software life cycle that ensures consistency and repeatability of process activities.
d. Provide a known point for review, assessing status, and change control by control of configuration items and the establishment of baselines.

e. Provide controls that ensure problems receive attention and changes are recorded, approved, and implemented.
f. Provide evidence of approval of the software by control of the outputs of the software life cycle processes.
g. Aid the assessment of the software product compliance with requirements.
h. Ensure that secure physical archiving, recovery and control are maintained for the configuration items.

The objectives for SCM are independent of software level. However, two categories of software life cycle data may exist based on the SCM controls applied to the data.

2.19 Software Quality Assurance Process

The objectives and activities of the software quality assurance (SQA) process. The SQA process is applied as defined by the software planning process and the Software Quality Assurance Plan. Outputs of the SQA process activities are recorded in Software Quality Assurance Records or other software life cycle data.

The SQA process assesses the software life cycle processes and their outputs to obtain assurance that the objectives are satisfied, that deficiencies are detected, evaluated, tracked and resolved, and that the software product and software life cycle data conform to certification requirements.

2.19.1 Software Quality Assurance Process Objectives

The SQA process objectives provide confidence that the software life cycle processes produce software that conforms to its requirements by assuring that these processes are performed in compliance with the approved software plans and standards.

The objectives of the SQA process are to obtain assurance that:

a. Software development processes and integral processes comply with approved software plans and standards.
b. The transition criteria for the software life cycle processes are satisfied.
c. A conformity review of the software product is conducted.

2.19.2 Software Quality Assurance Process Activities

a. The SQA process should take an active role in the activities of the software life cycle processes, and have those performing the SQA process enabled with the authority, responsibility and independence to ensure that the SQA process objectives are satisfied.

b. The SQA process should provide assurance that software plans and standards are developed and reviewed for consistency.

c. The SQA process should provide assurance that the software life cycle processes comply with the approved software plans and standards.

d. The SQA process should include audits of the software development and integral processes during the software life cycle to obtain assurance that:

   1. Software plans are available as specified.
   2. Deviations from the software plans and standards are detected, recorded, evaluated,tracked and resolved.
   3. Approved deviations are recorded.
   4. The software development environment has been provided as specified in the softwareplans.
   5. The problem reporting, tracking and corrective action process complies with the Software Configuration Management Plan.
   6. Inputs provided to the software life cycle processes by the on-going system safetyassessment process have been addressed.

e. The SQA process should provide assurance that the transition criteria for the software life cycle processes have been satisfied in compliance with the approved software plans.

f. The SQA process should provide assurance that software life cycle data is controlled in accordance with the control categories.

g. Prior to the delivery of software products submitted as part of a certification application, a software conformity review should be conducted.

h. The SQA process should produce records of the SQA process activities including audit results and evidence of completion of the software conformity review for each software product submitted as part of certification application.

3 RTCA/DO- 178C

RTCA/DO-178C, also known simply as DO-178C, is a set of guidelines and standards for the development and certification of airborne software. It is the most recent and current version of the DO-178 series of standards. DO-178C was published by the Radio Technical Commission for Aeronautics (RTCA), in collaboration with EUROCAE, and it supersedes its predecessor, DO-178B.

Key features and aspects of RTCA/DO-178C include:

1. **Modernization:** DO-178C modernizes the guidance to align with contemporary software engineering practices. It takes into account advances in software development, including object-oriented design, model-based development, and formal methods.
2. **Software Levels:** DO-178C classifies software into different levels of criticality, ranging from Level A (most critical) to Level E (least critical). This classification helps in determining the rigor of the certification process and the level of testing and verification required for each software component.
3. **Requirements-Based Approach:** DO-178C places a strong emphasis on a requirements-based approach. It requires a clear and traceable set of requirements for each software component. The software development process must ensure that the software requirements are met.
4. **Tool Qualification:** DO-178C provides guidelines for the qualification of software development and verification tools. It helps ensure that the tools used in the development process are reliable and do not introduce errors.
5. **Model-Based Development:** The standard encourages the use of model-based development approaches, where software requirements and design are represented using formal models. This approach can enhance the clarity and precision of the software development process.
6. **Verification and Validation:** DO-178C outlines detailed guidelines for verification and validation processes, which include various forms of testing, reviews, and

analysis to demonstrate that the software functions correctly and safely.

7. **Certification Considerations:** It provides guidance on the certification process and documentation required for gaining approval from aviation authorities like the Federal Aviation Administration (FAA) in the United States.
8. **Supplement and Appendices:** DO-178C includes several supplements and appendices that offer additional information and guidance for specific topics or technologies, such as tool qualification and model-based development.

DO-178C is essential for the aviation industry to ensure the safety and reliability of software in airborne systems, including aircraft avionics. Compliance with this standard is crucial for software developers and system integrators, as it helps in obtaining certification from aviation authorities, which is necessary for deploying software in critical aviation systems.

3.1 RTCA/DO- 178C process and Objectives

The procedure for complying with RTCA/DO-178C involves several steps to ensure the development and certification of airborne software in compliance with the standard. Here's an overview of the general procedure:

1. **Project Planning:**
   - Define the scope of the project and identify the software's criticality level (Level A to E).
   - Establish a plan that outlines the software development and verification processes, including objectives, requirements, schedules, and resources.
2. **Software Requirements:**
   - Develop clear, unambiguous, and verifiable software requirements.
   - Ensure that the requirements are traceable throughout the development process.
3. **Software Design:**
   - Create a detailed software design that maps to the requirements.
   - Apply modern software engineering practices, such as model-based development or formal methods if applicable.
4. **Coding and Unit Testing:**
   - Write the code based on the design.

- Perform unit testing to ensure that each software unit meets its requirements.

5. **Software Integration and Testing:**
    - Integrate the individual software units into a complete system.
    - Conduct integration testing to verify the interaction of software components and their compliance with requirements.
6. **Software Verification:**
    - Perform various forms of software verification, including reviews, static analysis, dynamic analysis, and formal methods, depending on the software's criticality level.
    - Ensure that verification activities are planned, documented, and traceable.
7. **Validation Testing:**
    - Execute validation testing to demonstrate that the software functions correctly and safely within the context of the aircraft or system.
    - Validation testing may include system-level testing, simulation, and flight testing.
8. **Certification Process:**
    - Prepare comprehensive documentation, including certification artifacts, to support the certification process.
    - Collaborate with aviation authorities, such as the Federal Aviation Administration (FAA), to gain certification approval.
9. **Tool Qualification:**
    - Qualify any software development and verification tools used in the development process.
    - Ensure that tools do not introduce errors and are reliable.
10. **Configuration Management:**
    - Implement a robust configuration management process to track changes and versions of the software and associated documentation.
11. **Change Control:**
    - Implement a change control process to manage changes to software requirements, design, and code.
12. **Quality Assurance:**

- Establish a quality assurance process to monitor and audit the software development activities to ensure compliance with DO-178C.

13. Documentation and Traceability:

- Maintain thorough documentation throughout the software development and verification processes.
- Ensure traceability between requirements, design, code, and verification activities.

14. Review and Audit:

- Conduct periodic reviews and audits of the software development process to identify and address any issues or non-compliance.

15. Supplement and Appendices:

- Refer to any relevant supplements and appendices in DO-178C for additional guidance on specific topics or technologies.

It's important to note that DO-178C is a complex and detailed standard, and compliance with it requires careful planning, execution, and documentation. Organizations involved in airborne software development should have a structured and disciplined approach to meet the standard's requirements and gain certification from aviation authorities. The specific procedures and details can vary depending on the software's criticality level and the context of the project.

4 Difference between RTCA/DO- 178B and RTCA/DO- 178C

RTCA/DO-178B and RTCA/DO-178C are both sets of guidelines and standards for the development and certification of airborne software, but there are significant differences between the two versions. RTCA/DO-178C is an updated and more comprehensive version of the standard, which supersedes DO-178B. Here are some of the key differences between RTCA/DO-178B and RTCA/DO-178C:

1. **Modernization and Alignment with Contemporary Practices:**
    - DO-178B was published in 1992 and reflects the state of software engineering at that time. DO-178C, on the other hand, was published in 2011 and takes into account advances in software engineering practices and technologies,

including object-oriented design, model-based development, and formal methods.

2. **Software Levels and Classification:**
    - Both versions classify software into different levels of criticality (Level A to E). However, DO-178C refines and provides more detailed guidance on how to assign levels and the associated certification processes, making it more precise and clear.
3. **Requirements-Based Approach:**
    - Both versions emphasize a requirements-based approach, but DO-178C provides additional guidance and clarification on requirements management and traceability.
4. **Model-Based Development:**
    - DO-178C encourages the use of model-based development approaches, which was not as explicitly addressed in DO-178B. Model-based development can enhance the clarity and precision of the software development process.
5. **Tool Qualification:**
    - DO-178C provides more detailed guidelines for the qualification of software development and verification tools, recognizing the importance of ensuring the reliability of tools used in the development process.
6. **Certification Process and Documentation:**
    - DO-178C outlines a more comprehensive certification process and includes updated and expanded documentation requirements. It provides clearer guidance on how to obtain certification from aviation authorities.
7. **Integration with Other Standards:**
    - DO-178C aligns more closely with other related aviation standards, such as DO-254 for hardware considerations and DO-278 for integrated modular avionics. This integration helps in addressing the overall system's safety and reliability.
8. **Supplements and Appendices:**

- DO-178C includes several supplements and appendices that offer additional information and guidance on specific topics or technologies, such as tool qualification and model-based development.

In summary, DO-178C is an updated and more comprehensive version of the standard, incorporating modern software engineering practices and technologies. It provides clearer and more detailed guidance on software development and certification, with a stronger emphasis on traceability, model-based development, and tool qualification. Compliance with DO-178C is crucial for the aviation industry to ensure the safety and reliability of software in airborne systems.

5 Conclusion

The Above RTCA/DO- 178 standards includes the description and difference between the RTCA/DO- 178A, RTCA/Do- 178B and RTCA/DO- 178C.