Frédéric Pothon
ACG Solutions

DO-178C/ED-12C versus DO-178B/ED-12B

Changes and Improvements

Sept 2012

Contributions and Reviews

Laurent Pomies
DOSoft-Consulting

Cyrille Comar
AdaCore

Hervé Delseny
Airbus

Ben Brosgol
AdaCore

Content

1. Purpose of this document

This document identifies all the changes in the new release DO-178C/ED-12C, explains their rationale, and highlights the impact of these changes on the various software processes.

Chapter 2 summarizes the background for the work on the new release and presents the “Terms of Reference” that established its scope.

Chapter 3 itemizes the various documents (“products”) that needed to be updated or prepared in connection with DO-178C/ED-12C.

Chapter 4, the main body of this document, describes the various changes in detail, arranged by category:

- 4.1: Errors and inconsistencies
- 4.2: Inconsistent terminology
- 4.3: Unclear descriptions
- 4.4: Hidden objectives
- 4.5: Gaps and new topics

Chapter 5 explains the role of the supplements and outlines the Tool Qualification document.

Chapter 6 describes the treatment of DO-248/ED-94, the DO-178B/ED-12B clarifications document.

Analyses of the other products associated with the DO-178C/ED-12C release are provided in separate documents:

- Model-Based Development supplement
- Formal Methods supplement
- Object Oriented Technologies and related techniques supplement
- Tool Qualification Document

2. DO-178C/ED-12C: Statement of work

Since 1992, the aviation industry and certification authorities around the world have used the considerations in DO-178B/ED-12B as an acceptable means of compliance for software approval in the certification of airborne systems and equipment. As experience was gained in the use of DO-178B/ED-12B, questions were raised regarding the document's content and its application. Some of these questions were addressed through the work of SC-190/WG-52 in the development of DO-248B/ED-94B. However, DO-248B/ED-94B does not contain additional guidance for use as compliance, but only clarifications. Additionally, advances in hardware and software technology resulted in software development methodologies and issues which were not adequately addressed in DO-178B/ED-12B or DO-248B/ED-94B.

In response to these developments, RTCA and EUROCAE formed a Software *Ad Hoc* committee to update the documents to meet several goals:

- Address emerging software trends and technologies
- Implement an approach that can change with the technology.
- Provide clear and consistent ties with the systems and safety processes.
- Rationalize the applicable guidance (from CAST papers, FAA orders, Certification Review Items (CRI), etc.) by incorporation into a consensus-approved document

Other improvements were proposed by industrial partners:

- Enhance the guidance for the use of COTS and alternate means,
- Propose a more appropriate tool qualification guidance and clarify the certification credit of a qualified tool,
- State that formal methods are an acceptable alternate means of compliance,
- Provide clear guidance on the use of model based development

Since DO-178B/ED-12B has struck an appropriate balance between the effort required to demonstrate compliance with its objectives and the resulting confidence in the correctness and safety of the software, there was no perceived need to make it more difficult to achieve compliance. Thus underpinning the revision was the desire to meet the goals and incorporate the improvements listed above "without raising the bar".

In December 2004, RTCA and EUROCAE approved the sponsorship of the joint Special Committee 205/Working Group 71, "SC-205/WG-71" (referred to simply as "SCWG" hereafter), to update the regulations on software used in airborne systems and equipment, with the following "Terms of reference" establishing the scope of the changes:

1. Modify DO-178B/ED-12B to become DO-178C/ED-12C, or other document number.

2. Modify DO-248B/ED-94B to become DO-248C/ED-94C, or other document number.

3. Consolidate Software Development Guidance.

4. Consolidate Software Development Guidelines.

5. Develop and document technology-specific or method-specific guidance and guidelines.

6. Determine, document and report the effects of DO-178C/ED-12C or other modified documents to DO-278/ED-109 and recommend direction to ensure consistency.

7. Develop and document the rationale for each DO-178B/ED-12B objective.

8. Evaluate the issues in the 'Software Issues List.xls' spreadsheet produced by the Software Ad Hoc committee and other identified issues. Determine 'if', 'where' and 'how' each issue should be addressed.

9. Coordinate SCWG products with software certification authorities via Certification Authorities Software Team (CAST) or other appropriate groups.

10. Coordinate with other groups and existing organizations (for example, SAE S18, WG-63, SC-200/WG-60), as appropriate.

11. Report to the SCWG's governing body the direction being taken by the committee within 6-9 months after the first SCWG meeting.

12. Work with RTCA and EUROCAE to explore and implement ways of expanding the usability of the deliverables (for example, hypertext electronic versions).

13. Modify DO-278/ED-109 to become DO-278A/ED-109A, or other document number.

14. Submit to RTCA and EUROCAE a DO-178C/ED-12C and DO-278A/ED-109A commonality analysis when documents are finalized.

It was also recognized that DO-178B/ED-12B's basic principles have demonstrated their relevance and value, and should remain unchanged. Directives were defined to accomplish the changes while not compromising these principles. Initially, one of these directives was that "the modifications should "*strive to minimize changes to the existing text (i.e., objectives, activities, software levels, and document structure)."*

Following many procedural discussions, this directive was rephrased. The objective is now to "*make those changes to the existing text that are needed to adequately address the current state of the art and practice in software development in support of system safety, to address emerging trends, and to allow change with technology".* This text better fits with the initial intent.

Overall, the complete set of directives was:

"1. Maintain the current objective-based approach for software assurance.

2. Maintain the technology independent nature of the DO-178B/ED-12B and DO-278/ED-109 objectives.

3. Evaluate issues as brought forth to the SCWG. For any candidate guidance modifications determine if the issue can be satisfied first in guideline-related documents.

4. Modifications to DO-178B/ED-12B and DO-278/ED-109 should:

> *i. In the context of maintaining backward compatibility with DO-178B/ED-12B, make those changes to the existing text that are needed to adequately address the current states of the art and practice in software development in support of system safety, to address emerging trends, and to allow change with technology.*

ii. Consider the economic impact relative to system certification or approval without compromising system safety.

iii. Address clear errors or inconsistencies in DO-178B/ED-12B and DO- 278/ED-109.

iv. Fill any clear gaps in DO-178B/ED-12B and DO-278/ED-109.

v. Meet a documented need to a defined assurance benefit.

vi. Report any proposed changes to the number of software levels or mapping of levels to hazard categories to the SCWG's governing body and provide a documented substantiated need, at the earliest feasible opportunity. Communicate back to the SCWG at large, any concerns of the governing body.

vii. Ensure that all deliverables produced by the committee contain consistent and complete usability mechanisms (for example, indexes, glossaries)."

3. DO-178C/ED-12C Products

In carrying out the directives listed in the previous chapter, *SCWG* ended up with seven documents to update or generate:

- **DO-178B/ED-12B** itself; the revised version is **DO-178C/ED-12C**.

- **DO-278/ED-109**: This document is applicable to ground-based systems (CNS and ATM software). This kind of software is not airborne software but may have an impact on safety. Before DO-278/ED-109, application of DO-178B/ED-12B was requested, but some ground software-specific needs had to be addressed, mainly the extensive use of COTS software. The need for specific guidelines and recommendations emerged before 2004. At that time, DO-178/ED-12 had not entered yet its modification process. Therefore a new specific document was created (DO-278/ED-109). It is inspired by DO-178/ED-12, and a large part of the document references DO-178B/ED-12B.

  The original intent of SCWG was to merge DO-278/ED-109 with DO-178/ED-12, and address all the ground-specific topics through either the "additional considerations" section or a dedicated supplement. This approach was rejected by RTCA/EUROCAE, in part because some industry representatives were concerned about possible side-effects on in-progress projects using DO-278/ED-109.. Instead, DO-278/ED-109 was revised as a separate activity: most of the changes had the goal of consistency with DO-178C/ED-12C, but the document itself is independent from DO-178C/ED-12C. The revised version is **DO-278A/ED-109A**.

- **DO-248B/ED-94B:** This document provides clarification of DO-178B/ED-12B. An update was necessary for several reasons. First, some information became obsolete or was incorporated into the core document. Furthermore, information had to be added to deal with new topics in DO-178C/ED-12C. The scope of the new release was also extended to the ground based domain (DO-278A/ED-109A): The related changes mainly consisted in adding references to both DO-178 and DO-278, or limiting the applicability of a piece of text to one of the two domains. The revised version is **DO-248C/ED-94C**.

  DO-248C/ED-94C now contains rationale for each DO-178C/ED-12C objective. The initial intent was to complete this rationale in the early phase of the SCWG work plan, which would have helped resolve some of the debates around DO-178/ED-12 core text changes. Unfortunately, the text was not produced early enough for this purpose. Nevertheless the rationale text is part of DO-248C/ED-94C and may help readers properly understand the contents of DO-178C/ED-12C (and its supplements).

- **Three supplements:** A basic principle of DO-178C/ED-12C and DO-278A/ED-109A is to be technology independent**.** The current state of the art in software engineering clearly includes techniques that are useful in developing airborne or ground-based systems and thus should be addressed by the SCWG, but expanding the core text of the two documents would not have been a practical approach. Instead, SCWG recommended preparing one or more supplements to address several new specific techniques. Used in conjunction with DO-178C/ED-12C and DO-278A/ED-109A, these supplements would amend the guidance to account for the new software technologies.

  A "supplement" supplements the guidance given in DO-178C/ED-12C and DO-278A/ED-109A. Each of them may be used in conjunction with any other supplement. A supplement identifies the additions, modifications, and deletions to DO-178C/ED-12C or DO-278A/ED-

109A objectives when a specific technique or method is used as part of a software life cycle, and provides any additional guidance required.

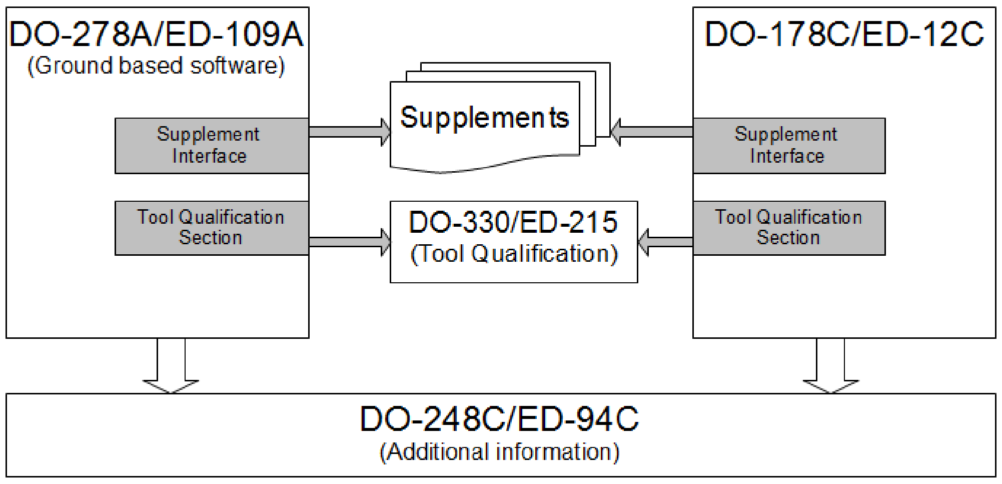
In the scope of this SCWG, three supplements were developed:

- **DO-331/ED-216**: *Model Based Development and Verification* supplement
- **DO-332/ED-217**: *Object Oriented Technology and Related Techniques* supplement
- **DO-333/ED-218**: *Formal Methods* supplement

In principle, additional supplements may be developed later without changing the core documents or any of the existing supplements.

- **Tool Qualification Document:** The Tool Qualification guidance in DO-178B/ED-12B had to be revised, as it was deemed unnecessarily difficult to apply and not sufficiently detailed to address tool specifics . The nature of the guidance to be provided for tool qualification does not fit with the concept of a supplement, since it not only amends the core guidance but also constitutes a complete and standalone set of recommendations, objectives, and guidance. In addition, it was recognized that the guidance for qualifying a tool should not be limited to the airborne domain. Based on these considerations, a completely new document was developed for tool qualification, and is domain independent. This document, **DO-330/ED-215**, may be referenced by other regulatory text, e.g. related to ground-based software or complex electronic hardware, and may also be used by other domains such as space, automotive, or medical systems.

The following figure displays the relationships among the different products:



4. Description of the changes

4.1 Errors and Inconsistencies

DO-178C/ED-12C corrects DO-178B/ED-12B's known errors and inconsistencies, most of which were considered as typographical errors. Some of these were already identified in DO-248B/ED-94B.

Here are the most significant corrections:

4.1.1- Error in Table A-7

In Objective 1 ("Test procedures are correct"), the output is incorrectly identified as "software verification cases and procedures". To satisfy this objective, the "Software Verification Cases and Procedures" need to be used as inputs to a verification activity. This activity (analysis and/or review, potentially tool based) produces an output, categorized as "Software Verification Results". In the table, the output of Objective 1 has thus been replaced with "software verification results".

The reference column was also updated (11.13 replaced with 11.14)

4.1.2- Errors in cross references

- In Chapter 7 (Configuration Management Process), Table 7-1 identifies the SCM Process Objectives associated with CC1 and CC2 data. The correct reference for "protection against unauthorized changes" is 7.2.7b(1) (7.2.7.b.1 with new numbering format) and not 7.2.5b(1)

- In Table A-4, Objective 9 ("Software architecture is consistent"), the correct reference of the objective is 6.3.3.b, not 6.3.2.b

4.1.3- Testing: process or activity?

Since verification is a combination of analyses, reviews and test, the term "testing process" is incorrect.

- The title of Section 6.4 was changed from "Software Testing Process" to "Software Testing"
- The title of Figure 6.1 was changed from "SOFTWARE TESTING PROCESS" to "SOFTWARE TESTING ACTIVITIES"
- In section 6.4, 2nd paragraph, the word "process" was replaced with "activity". The text now reads: "Figure 6-1 is a diagram of the software testing *activities…"*.

4.1.4- Partitioning integrity

ED-12B/DO-178B Section 6.3.3 deals with verification of the Software Architecture. In particular, item 'f' addresses the integrity of the partitioning. The original text was: "*The objective is to ensure that partitioning breaches are prevented or isolated*".

For some time the phrase "or isolated" at the end of this sentence has been known to be inappropriate. Errata #5 from ED-94B/DO-248B thus suggests deleting this phrase from the sentence, resulting in the following wording: “The objective is to ensure that partitioning breaches are prevented.” This change has been made in ED-12C/DO-178C.

Here is some background / rationale on this issue.

Preventing partition breaches is important, and entails somehow verifying the efficiency of the partitioning strategy. It is essential to identify and track the potential conditions that would lead the actual partitioning mechanism to fail. Once this is done, a workaround solution can be set up, or the partitioning mechanism can be revised.

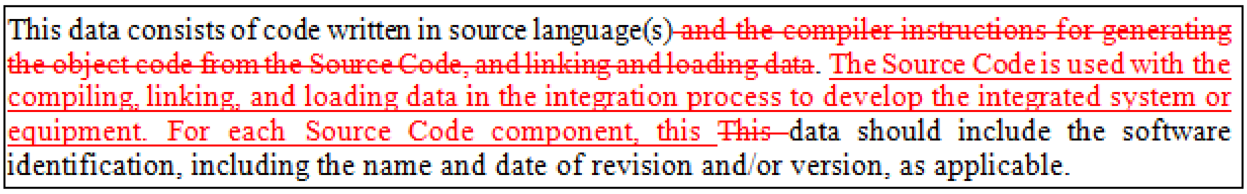
Using the word “isolate” created confusion. In the context of partitioning, the notion of isolation has a specialized meaning, referring to segregation between functionalities. However, its usage in DO-178B/ED-12B Section 6.3.3 had a different interpretation. As originally phrased, it appeared that partitioning breaches should be addressed as acceptable system states, and that is definitely incorrect. A good segregation strategy leads to the absence of partitioning breaches, not their isolation. Removing the “or isolated” wording solves this problem.

4.1.5- Source code and Object code

In its discussion of the Software Coding Process and the Integration Process, DO-178B/ED-12B was sometimes unclear and inconsistent with respect to the meanings of Source Code, generation of object code, and “linking and loading data”.

- One problem is that the definition of Source Code in Section 11.11 includes the compiler instructions (i.e., the commands / options for invoking the compiler) and the linking and loading data. There is no reason to consider these as part of the Source Code. In addition, the concept of “linking and loading data” is not well-defined.

The definition of Source Code has been corrected in the glossary to DO-178C/ED-12C:


This data consists of code written in source language(s) ~~and the compiler instructions for generating the object code from the Source Code, and linking and loading data~~. The Source Code is used with the compiling, linking, and loading data in the integration process to develop the integrated system or equipment. For each Source Code component, this ~~This~~ data should include the software identification, including the name and date of revision and/or version, as applicable.

- The last part of the definition requires each Source Code component to be clearly identified. This identification is not limited to the component name, but also specifies the version. This is unchanged from DO-178B/ED-12B, except that now it has been made explicit that it is applicable to “each source component”.
- In addition to the definition of “Source Code”, there were some issues with the DO-1787B/ED-12B core text in Section 5.3.2: “The primary results of this [software coding] process are Source Code … and object code”. The reference to object code implies that compilation has to be performed as part of the coding process. However, nothing in the list of the software coding process activities relates to compilation. Likewise, the verification section (6.3.4) addressing the outputs of the software coding process does not provide any recommendation for the verification of the object code. To correct this inconsistency, Section 5.3.2 has been modified in DO-178C/ED-12C to eliminate object code as a result of the software coding process. The text now reads: “The primary result of this [software coding] process … is the Source Code”.

- Additional changes were made in both the Software Coding Process and Integration Process sections to clarify that object code is not an output of the software coding process but rather an intermediate product of the integration process:

§5.3

> Note: For the purpose of this document, compiling, linking, and loading are dealt with under the Integration Process (see 5.4).

§5.4.2

> The outputs of the integration process are the object code, Executable Object Code (see 11.12), Parameter Data Item File (see 11.22), and the compiling, linking, and loading data. The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied.

- In addition to clarifying the text of the development processes, DO-178C/ED-12C has improved the wording of the verification process. To address the original absence of recommendations concerning the outputs of the compiler, the section related to verification of the outputs of the integration process (§6.3.5) now includes an activity for the detailed examination of compiler warnings.

In spite of the significant improvements on this topic as detailed above, some errors/gaps from DO18B/ED-12B remain in DO-178C/ED-12C:

- “Object code” is still not identified as a software life cycle data item. This is not as significant an issue as in DO-178B/ED-12B, because object code is no longer an interface between two processes (Coding/Integration) but only an intermediate result of the integration process. No transition criteria should be based on the generation of object code.
- The term “Compiling, linking and loading data” are inconsistently used: They sometimes refer to the commands and options used in invoking the tools, and sometimes the results of the compiling, linking and loading activities. The most confusing text can be found in Section 5.4.2 where “compiling, linking and loading data” are identified as both an input (bullet ‘a’) and an output (third paragraph) of the integration process.

Despite these problems, the text is understandable. However, it is recommended that the software life cycle description (and thus the plans) be clear about the difference between the “instructions” and “results” of compiling, linking, and loading. The use of the general word “data” should not be used, to avoid any confusion between the instructions and results.

4.1.6- Control Category of Design Description (level C)

In DO-178B/ED-12B table A-2 the Design Description’s Control Category is specified as CC1 for software levels A and B, but only CC2 for software levels C and D.

However, as defined in §9.4, the Design Description is part of the Software Life Cycle Data Related to Type Design. The Control Category for all of this data is CC1 for all software levels – with the sole exception of the Design Description, which is at CC2 for SW levels C and D as noted above.

The application of CC2 to the Design Description for software at level C was considered as an error, because

[1] Low Level Requirements are a part of the Design Description,

[2] Source Code is based on the Low Level Requirements, and

[3] Source Code for SW level C is controlled at CC1.

Since Source Code is controlled at CC1, the Design Description (which includes Low Level Requirements) should also be controlled at CC1.

Note that this change is not applicable to level D as explained in §5.5.

4.2 Inconsistent Terminology

DO-178C/ ED-12C clarifies the use of specific terms such as “guidance,” “guidelines,” “purpose,” “goal,” “objective,” and “activity”. The glossary was revised and the core text modified whenever needed. The use of these specific terms is now consistent throughout the document.

These wording improvements do not affect the original meaning of the text.

4.2.1 “Guidance” and “Guidelines”

Beyond their inconsistent usage throughout the document, the real meaning of these terms was confusing. (They were not part of the DO-178B/ED-12B glossary. They are still not defined in the new glossary but, as will be seen below, the revisions to the text have cleared up the confusion.)

Since “guidance” conveys a slightly stronger sense of obligation than “guidelines”, the SCWG decided to use the term “guidance” for all the pieces of text that are considered as actual “recommendations” .To avoid confusion, it was also decided to replace the term “guidelines” (widely used in DO-178B/ED-12B) with “supporting information”, whenever the text was more “information” oriented than “recommendation” oriented. These were cases where the primary intent was to help the reader to understand the context or the text itself. Hence, all the “notes” included in the text are not guidance. Also the complete DO-248/ED-94 document falls into the “supporting information” category, and not guidance.

In summary, most of the occurrences of “guidelines” were replaced by “guidance”, and the others by “supporting information”.

Though the glossary does not include definitions for the terms “guidance” and “supporting information”, the core text section 1.1 (purpose) clearly explains these two concepts:

§1.1

This guidance includes:

- Objectives for software life cycle processes.
- Activities that provide a means for satisfying those objectives.
- Descriptions of the evidence in the form of software life cycle data that indicate that the objectives have been satisfied.
- Variations in the objectives, independence, software life cycle data, and control categories by software level.
- Additional considerations (for example, previously developed software) that are applicable to certain applications.
- Definition of terms provided in the glossary.

In addition to guidance, supporting information is provided to assist the reader’s understanding.

4.2.2 Objectives and activities

Within the aerospace industry, DO-178B/ED-12B has often been read and understood as simply a collection of objectives, with little consideration to the wide set of activities described in the text.

The activities description is part of the guidance and provides a proposed means (with a long and successful track record in practice) to satisfy the objectives. It was considered as important to have a clearer identification of these activities, and a consistency link between objectives and proposed activities. Therefore, some modifications were incorporated in DO-178C/ED-12C to emphasize that the complete text – activities as well as objectives – should be considered.

For example, Section 1.4, titled “How to Use This Document”, reinforces the point that activities are a major part of the overall guidance. Hence, while the Annex A tables in DO-178B/ED-12B refer only to the objectives, they now also include references to each activity.

Accordingly, a specific review of DO-178B/ED-12B was performed in order to assess the completeness and consistency of the objectives and activities identification. The following items explain the main resulting modifications.

4.2.2.1- New activity in Software planning process

One objective of the software planning process (Section 4.1) is to address the additional considerations such as those described in Section 12. However, in the Planning Process Activities section (4.2), there was no activity related to the satisfaction of this objective.

A new bullet was thus added: §4.2.k

> The software planning process should address any additional considerations that are applicable.

4.2.2.2- New activity in Design process

The primary objective of the Design process is to develop the architecture and the Low-level requirements. Control and Data flow are part of the architecture, but none of the Design Process activities required developing this subset of the design data.

A new activity was thus added: §5.2.2.d

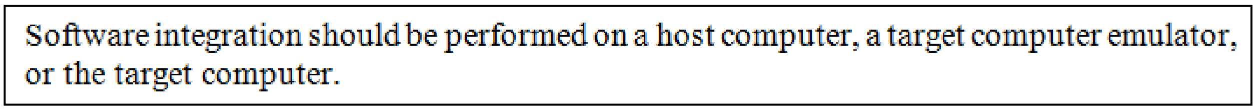
> Interfaces between software components, in the form of data flow and control flow, should be defined to be consistent between the components.

4.2.2.3- Changes in integration process activity description

The real purpose of DO-178B/ED-12B Section 5.4.3 (“Integration Considerations”), namely the production of the executable object code, was not clear. This goal is now explicitly identified in DO-178C/ED-12C, and the text in 5.4.3 (if not moved elsewhere as explained below) is now merged into 5.4.2 as Integration Process Activities.

Considerations related to deactivated code and patches were included in the old section 5.4.3. However, deactivated code is not really related to the integration issue and thus is now described in a specific new section, “Designing for Deactivated Code” (5.2.4), in the Software Design Process section of DO-178C/ED-12C. Considerations related to patches are now identified as activities in the Integration Process Activities section (5.4.2).

In addition, Section 5.4.2 now includes a new bullet item to clearly specify that multiple environments may be used to perform software integration: §5.4.2.b



Software integration should be performed on a host computer, a target computer emulator, or the target computer.

4.2.2.4- Verification process chapter reorganization

Although the titles of DO-178B/ED-12B Sections 6.1 and 6.2 mention “Objectives” and “Activities”, respectively, these sections do not describe all the objectives and activities of the Verification Process. Many objectives and activities are documented elsewhere:

- Chapter 6 provides a set of subsections, each treating the verification of a specific output of the development processes and listing the objectives associated with the verification. For example, Section 6.3.1 deals with the verification of the High Level Requirements and identifies the objectives that must be met. Likewise, the LLR verification objectives are described in section 6.3.2.
- The description of the Verification Process Activities in Section 6.2 is rather general. Specific activities are presented in the subsections dealing with “reviews and analyses” and in the “Software Testing Process” section (6.4). The latter section includes specific analyses, such as test coverage analysis, in addition to testing.

In DO-178C/ED-12C, the titles of sections 6.1 and 6.2 were updated to make them consistent with the text. They are now “Purpose of Software Verification” and “Overview of Software Verification Process Activities”, respectively. Beyond this, no major change was made with regard to the “objectives and activities” consistency issue.

The correspondence between the sections in Chapter 6 of DO-178B/ED-12B and DO-178B/ED-12B is shown in the figure below:

DO178B/ED12B	DO178C/ED12C
6 Software verification process	**6 Software verification process**
6.1 Software Verification process Objectives	6.1 Purpose of Software Verification process
6.2 Software Verification Process Activities	6.2 Overview of Software Verif Process Activities
6.3 Software Reviews and analyses	6.3 Software Reviews and analyses
6.3.1 Reviews and analyses of HLR	6.3.1 Reviews and analyses of HLR
6.3.2 Reviews and analyses of LLR	6.3.2 Reviews and analyses of LLR
6.3.3 Reviews and analyses of Architecture	6.3.3 Reviews and analyses of Architecture
6.3.4 Reviews and analyses of Source Code	6.3.4 Reviews and analyses of Source Code
6.3.5 Reviews and analyses of outpts of Integration	6.3.5 Reviews and analyses of outpts of Integration
6.3.6 Reviews and analyses of tests *data*	*Switched to 6.4.5*
6.4 Software testing	6.4 Software testing
6.4.1 Test environment	6.4.1 Test environment
6.4.2 Requirement based test case selection	6.4.2 Requirement based test selection
6.4.3 Requirement Based testing methods	6.4.3 Requirement Based testing methods
6.4.4 Test coverage analysis	6.4.4 Test coverage analysis
	6.4.5 Reviews and analyses of tests *data*
	6.5 Software Verfication process Traceability

Note that there are significant changes in Chapter 6 for other reasons. An explanation for these changes will be given below.

4.2.2.5- Software Configuration Management Process objectives

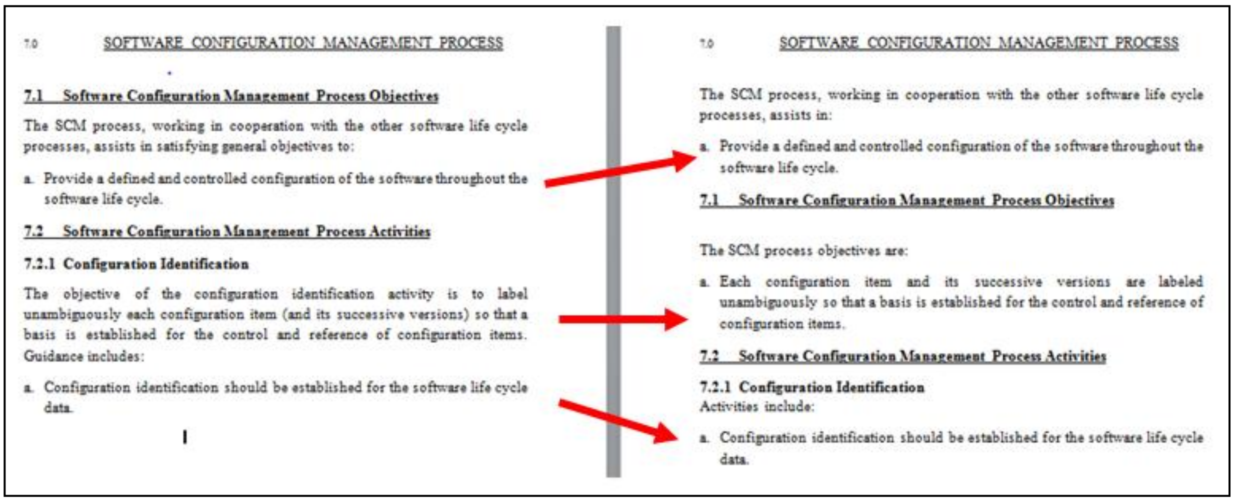
Section 7.1 of DO-178B/ED-12B, although titled "Software Configuration Management Process Objectives", did not clearly correspond to the objectives in Table A-8: The text of this section is now included in the introduction of Chapter 7 where it serves to present the general purpose of the SCM process.

DO-178C/ED-12C's section 7.1 consists of the material in the original 7.2 subsections that could be categorized as "objectives". The new 7.2 subsections are now phrased in terms of "Activities".

There is no new guidance in this section, but the text now makes the objective and activity descriptions consistent. Example:

DO-178B/ED-12B

DO-178C/ED612C



7.0 SOFTWARE CONFIGURATION MANAGEMENT PROCESS

7.1 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.

7.2 Software Configuration Management Process Activities

7.2.1 Configuration Identification

The objective of the configuration identification activity is to label unambiguously each configuration item (and its successive versions) so that a basis is established for the control and reference of configuration items. Guidance includes:

a. Configuration identification should be established for the software life cycle data.

7.0 SOFTWARE CONFIGURATION MANAGEMENT PROCESS

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

a. Provide a defined and controlled configuration of the software throughout the software life cycle.

7.1 Software Configuration Management Process Objectives

The SCM process objectives are:

a. Each configuration item and its successive versions are labeled unambiguously so that a basis is established for the control and reference of configuration items.

7.2 Software Configuration Management Process Activities

7.2.1 Configuration Identification
Activities include:

a. Configuration identification should be established for the software life cycle data.

4.2.2.6- Traceability to activities in the objective tables

One of the basic principles of DO-178B/ED-12B is that only objectives should be satisfied: the activities described in the text are to be regarded as a set of proposed means to satisfy the objectives. An applicant may choose activities other than those proposed, as long as evidence can be shown that the applicable objective(s) is/are satisfied.

This principle holds also in DO-178C/ED-12C. However, for the sake of completeness and to assist in the correct understanding of the document, the tables in Appendix A now provide (for each objective) the references to all sections where the suggested activities are listed, in addition to the references to the sections where the objective itself is described,

Here is an example:

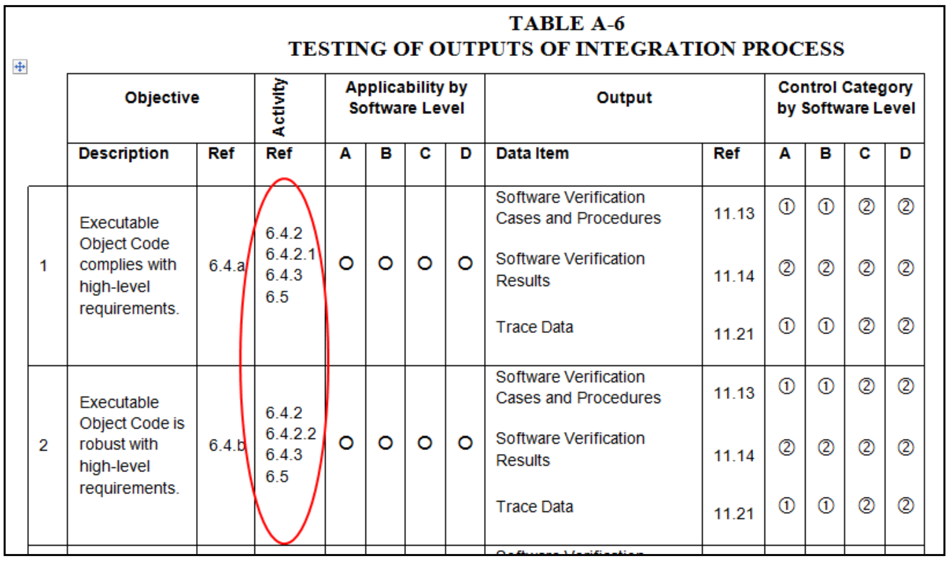


TABLE A-6
TESTING OF OUTPUTS OF INTEGRATION PROCESS

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Executable Object Code complies with high-level requirements.	6.4.a	6.4.2 6.4.2.1 6.4.3 6.5	○	○	○	○	Software Verification Cases and Procedures	11.13	①	①	②	②
								Software Verification Results	11.14	②	②	②	②
								Trace Data	11.21	①	①	②	②
2	Executable Object Code is robust with high-level requirements.	6.4.b	6.4.2 6.4.2.2 6.4.3 6.5	○	○	○	○	Software Verification Cases and Procedures	11.13	①	①	②	②
								Software Verification Results	11.14	②	②	②	②
								Trace Data	11.21	①	①	②	②

4.3 Unclear descriptions

DO-178C/ED-12C clarifies a number of concepts whose descriptions were incomplete or confusing in DO-178B/ED-12B

4.3.1 Coordinated System/Software Aspects

During the development of the DO-178B/ED-12B , no system level guidance was available. Therefore, Chapter 2 was written to address all potential system aspects to be considered at the software level. The main result was to identify the interfaces with system life cycle processes.

SAE/EUROCAE published the first release of its system level guidance document (ARP4754/ED-79) in November 1996. Subsequently ARP4754A/ED–79A was issued in December 2010, titled “*Guidelines for Development of Civil Aircraft and Systems*”.

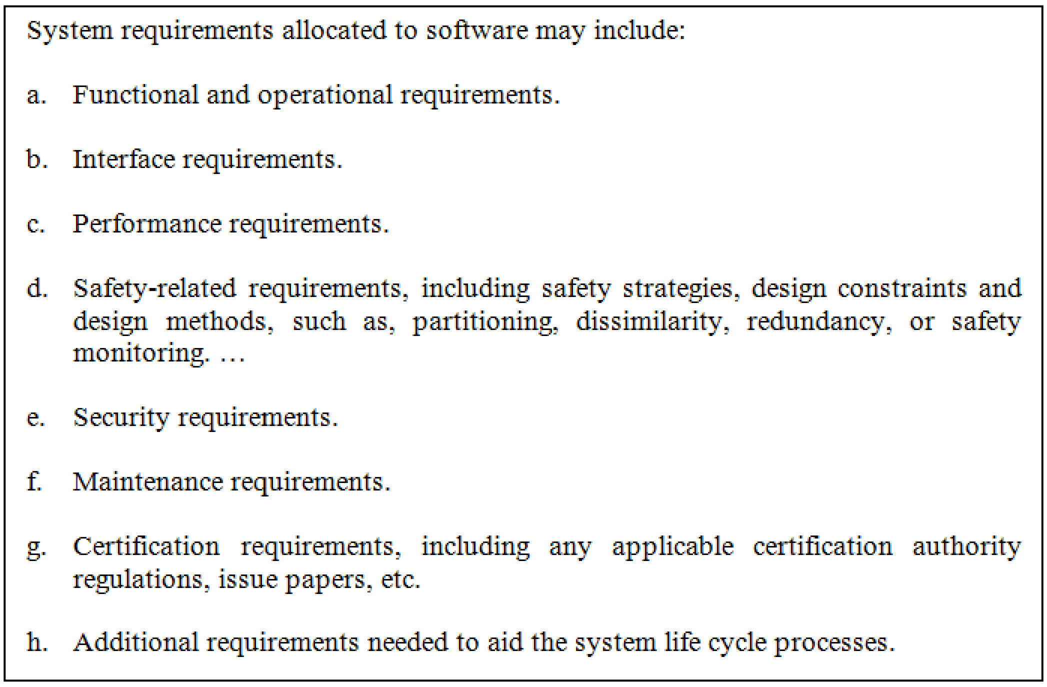
DO-178C/ED-12C incorporates significant changes in Chapter 2, most of which are based on text from ARP4754A/ED-79A. Regular coordination with the committee in charge of updating ARP4754/ED-79 was also established.

Just as in DO-178B/ED-12B, except for architectural considerations, this chapter is not part of the guidance material. It is provided for information only, and does not include any objectives.

The following points were clarified:

- **System requirements allocated to software**

The system requirements allocated to software are now described in a dedicated subsection (§2.1). The text mainly specifies which system requirements may be allocated to software:



System requirements allocated to software may include:

a. Functional and operational requirements.

b. Interface requirements.

c. Performance requirements.

d. Safety-related requirements, including safety strategies, design constraints and design methods, such as, partitioning, dissimilarity, redundancy, or safety monitoring. …

e. Security requirements.

f. Maintenance requirements.

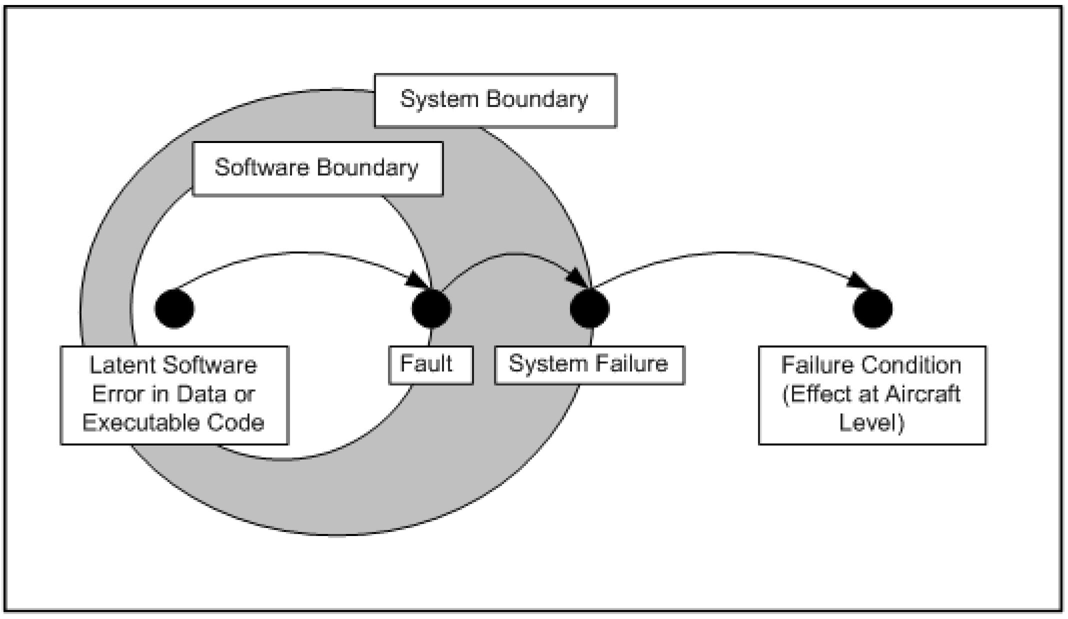
g. Certification requirements, including any applicable certification authority regulations, issue papers, etc.

h. Additional requirements needed to aid the system life cycle processes.

(Note the wording “may include” instead of “should include”, identifying that the information is not part of the guidance.)

- **Contribution of a software error to system failure / Failure Condition Categorization / software level**

The software level of a software component is based upon its contribution to potential failure conditions. The relationship between software errors and failure conditions is briefly addressed in Section 2.3.1 but does not introduce any new concepts compared to DO178B/ED-12B. The following figure is provided (Figure 2.2):


System Boundary
Software Boundary
Latent Software Error in Data or Executable Code
Fault
System Failure
Failure Condition (Effect at Aircraft Level)

Likewise, the concepts related to failure conditions categorization and software level definition / determination remain unchanged.

However, the definition of each failure condition was slightly updated for consistency with FAE/CS 25-1309

Catastrophic: Failure conditions which would ~~prevent continued safe flight and landing~~ result in multiple fatalities, usually with the loss of the airplane.

Hazardous~~/Severe-Major~~: Failure conditions which would reduce the capability of the ~~aircraft~~ airplane or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be:

(1) a large reduction in safety margins or functional capabilities,

(2) physical distress or ~~higher~~ excessive workload such that the flight crew ~~could not~~ cannot be relied ~~on~~ upon to perform their tasks accurately or completely, or

(3) ~~adverse effects on occupants including~~ serious or ~~potentially~~ fatal injuries to a relatively small number of ~~those~~ the occupants other than the flight crew.

Major: Failure conditions which would reduce the capability of the ~~aircraft~~ airplane or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to ~~occupants~~ flight crew, or physical distress to passengers or cabin crew, possibly including injuries.

Minor: Failure conditions which would not significantly reduce ~~aircraft~~ airplane safety, and which would involve crew actions that are well within their capabilities. Minor failure conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight increase in crew workload, such as, routine flight plan changes, or some ~~inconvenience to occupants~~ physical discomfort to passengers or cabin crew.

No Effect: Failure conditions ~~which do~~ that would have no effect on safety; for example, Failure Conditions that would not affect the operational capability of the ~~aircraft~~ airplane or increase crew workload.

- **Architectural considerations**

The DO-178C/ED-12C changes in this area can be categorized as clarifications. As in DO-178B/ED-12B, this section (§2.3) identifies three possible techniques: Partitioning, Multiple Version Dissimilar software, and Safety monitoring. Each potentially limits the impact of failures, helps detect failures, and provides acceptable system responses for their containment. The text related to Multiple Version Dissimilar software and Safety monitoring is unchanged. For partitioning, the guidance is now clearer and more detailed.

- **Software considerations in system life cycle processes**

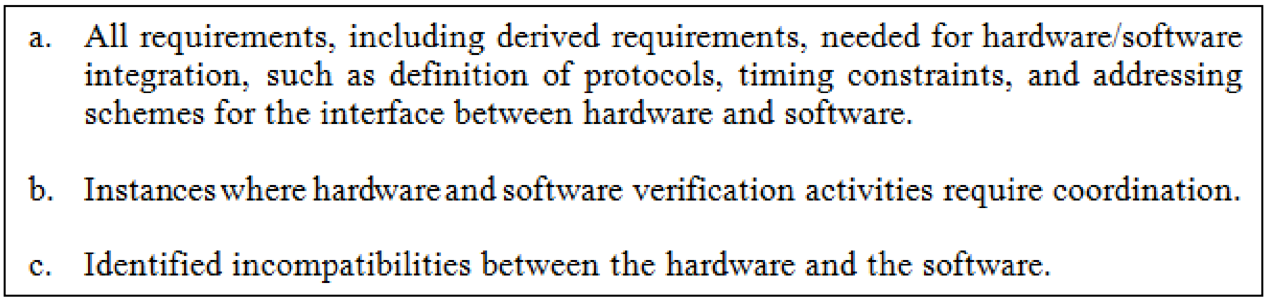
Section 2.5 has been reorganized to make the material more consistent. It deals with the following topics, which were already treated in DO-178B/ED-12B: User-modifiable software, COTS, Option-selectable software and Field-loadable software. It also adds a discussion of Parameter data items (see Parameter data item (PDI). Most of these topics are addressed in more detail in other sections. The added value of the text in Section 2 is to consider the possible impact of these technical decisions on the system processes.

- **Information flows**

Information flows between system processes and software processes are described**.** The text clearly specifies that *all* software derived requirements should be provided as input to the system processes, and not only those coming from the requirements process.

A new section was created for information flow between Software processes and hardware processes. No direct interfaces between hardware processes and software processes were identified in DO-178B/ED-12B.The revised approach is consistent with ARP4754/ED-79: "*This information should flow via the systems process*"; However, DO-178C/ED-12C is more flexible, as it proposes that the data be passed "*either as part of the system requirements allocation or during the development life cycle*."

§2.2.3: This information includes



a. All requirements, including derived requirements, needed for hardware/software integration, such as definition of protocols, timing constraints, and addressing schemes for the interface between hardware and software.

b. Instances where hardware and software verification activities require coordination.

c. Identified incompatibilities between the hardware and the software.

- **Interaction between system and software**

The possible credit taken from system activities to satisfy software objectives, or the possible application of software life cycle processes to satisfy system objectives, was specified in DO-1787B/ED-12B, but its description has now been clarified.

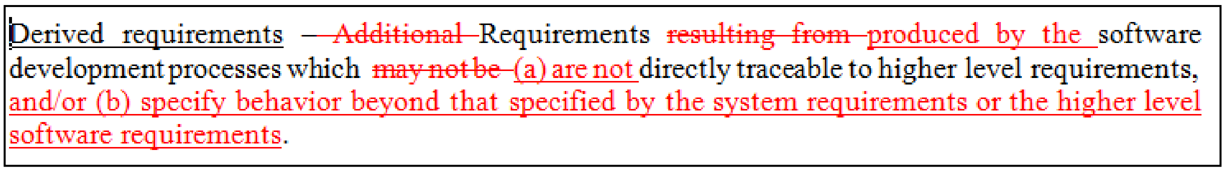
The information is presented in sections 2.6 and 2.5.6, and also in the description of the information flows:

- From system to software (§2.2.1): Identification of system activities to be performed as part of software processes, related acceptability evidence, and evidence of software verification activities performed by the system process.
- Form software to system (§2.2.2): Details of software verification activities proposed to be performed during system verification.

4.3.2 Derived requirements and traceability

Clarifications on this topic led to several changes throughout the document

First of all, the definition of derived requirements was updated, the focus being more on the content of the requirements rather than on the traceability aspects:



Derived requirements – ~~Additional~~ Requirements ~~resulting from~~ produced by the software development processes which ~~may not be~~ (a) are not directly traceable to higher level requirements, and/or (b) specify behavior beyond that specified by the system requirements or the higher level software requirements.

It is now considered that some “traceable” requirements can be identified as derived because they specify behavior beyond that specified in higher level of requirements. It should be noted that this does not really change the previous definition, as the term “may be not traceable” already opened the door to the same interpretation. FAQ#36 of DO-248C/ED-94C was reworked to provide examples of the two “classes” of derived requirements.

A correct application of this concept (derived requirements) requires good experience and maturity within the software engineering team: The purpose of the traceability feature is both to enable the verification of the complete implementation of the higher level of requirements and give visibility on the derived requirements, as now clarified in section §5.5 (new). Therefore, beyond the accurate definition/identification of the derived requirements, it is very important to define a traceability approach that actually supports and complies with the above purpose.

Likewise, section §6.5 (new) provides the purpose of the traceability, as long as verification data are concerned:

- between requirements and test cases, the purpose is to support the requirement coverage analysis
- between test cases and test procedures, the purpose is to enable the verification of the complete implementation of test cases into test procedures
- and finally between the test procedures and the test results, the purpose is to verify that the complete set of test procedures has been executed.

From a pure traceability standpoint, one of the main changes is to consider the "trace data" as new software life cycle data. However, what these trace data should look like is not specified, as their definition in the glossary allows multiple formats to be used: The traceability linkages may be shown with different techniques, as shown below:

> Trace data – Data providing evidence of traceability of development and verification processes' software life cycle data without implying the production of any particular artifact. Trace data may show linkages, for example, through the use of naming conventions or through the use of references or pointers either embedded in or external to the software life cycle data.

Trace data should be used wherever it is necessary to establish an association between two life cycle data items. The new wording requires bi-directionality of this association: §11.21

> Trace Data should be provided that demonstrates bi-directional associations between:
>
> a. System requirements allocated to software and high-level requirements.
>
> b. High-level requirements and low-level requirements.
>
> c. Low-level requirements and Source Code.
>
> d. Software Requirements and test cases.
>
> e. Test cases and test procedures.
>
> f. Test procedures and test results.

Another clarification is to explicitly require that rationale for derived requirements be provided during the development process. The wording "reason for their existence" has been added in the activity description, in addition to the analysis. This rationale should also be passed to the system process together with the requirements themselves. Since the analysis needs to be focused on a possible impact on safety, it was clarified that the system safety assessment process is part of the analysis of derived requirement. The objectives in table A-2 becomes "Derived HLR/LLR are defined and provided to the system processes, including the system safety assessment process".

4.3.3 User modifiable software

The new section 2.5 addresses the "Software Considerations in the System Life Cycle Processes", merging previous DO-178B/ED-12B sections 2.4, 2.5 and 2.7. Then, each subsection addresses a specific topic: user-modifiable software is addressed in section 2.5.2.

"User-modifiable software" was already covered within DO-178B/ED-12B, but guidance on this topic is enhanced in DO-178C/ED-12C, beginning by including its definition in the glossary.

> User-modifiable software – Software intended for modification without review by the certification authority, the airframe manufacturer, or the equipment vendor, if within the modification constraints established during the original certification project.

This definition establishes the scope of this guidance: the approach and constraints for user-modifiable software have to be defined and agreed prior to the first certification. Then the modifications performed in the scope of these constraints are not subject to any further authority review.

Within section 2.5.2, additional guidance is provided, with emphasis on considerations on safety impact (bullet a), and on the need for the applicant to provide all necessary information to the user (bullet f).

No change in the planning process section: As previously, specific planning process activities are expected, and the strategy for user modifiable software needs to be documented in the PSAC, within the additional considerations section.

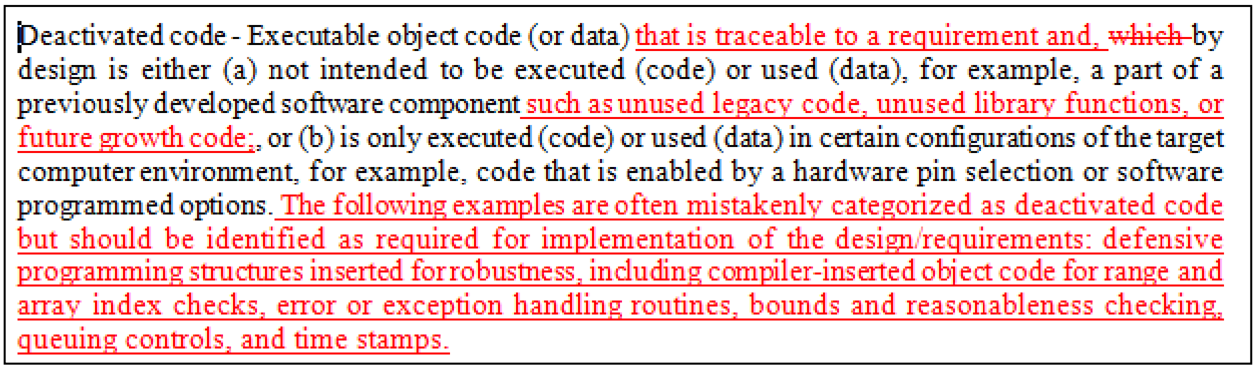
The design process still includes a section on “Designing for User-Modifiable Software” (§5.2.3). Additional guidance is provided to address the protection mechanisms. These mechanisms should be developed at the same software level as the non-modifiable components (if implemented in software). If a tool is involved in the protection mechanisms, it should be subject to tool qualification.

4.3.4 Deactivated code

Deactivated code was mainly addressed in DO-178B/ED-12B as something to be considered during the integration process and while resolving the structural coverage analysis issues. This was not fully consistent as the “*means to ensure that deactivated code cannot be enabled in the target computer*” was expected as part of the design data (§11.10), while no guidance was provided in the Design Process.

An approach leading to keep some deactivated code in the final software should be considered much earlier in the project, during the planning process of course, and then in the early phases of the development processes.

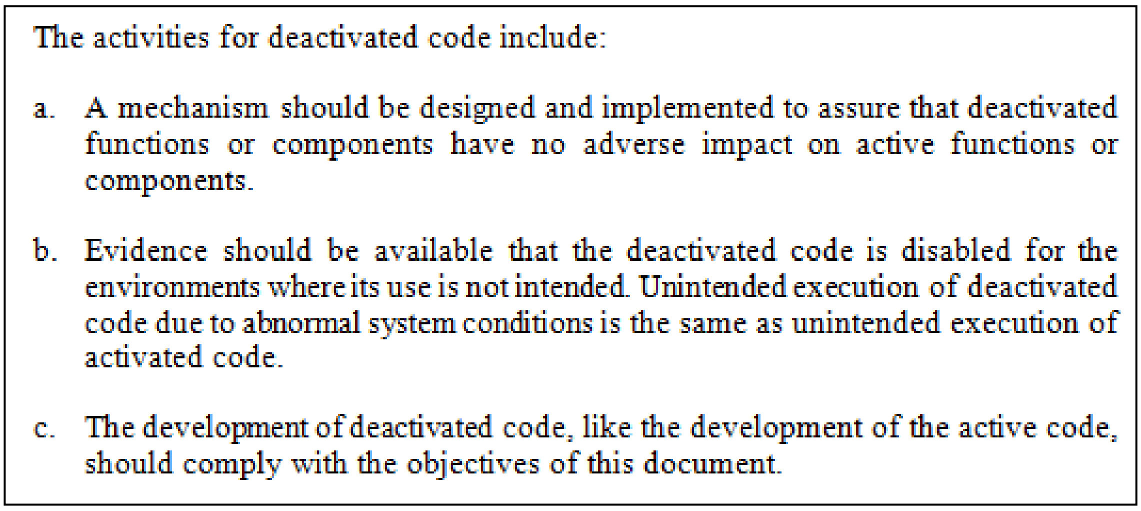
As a result, a new approach is introduced in DO-178C/ED-12C, starting with an enhancement of the definition, providing examples of what is and what is not deactivated code. The definition also clearly states that the deactivated code is really intentional, as it is traceable to requirements.


Deactivated code - Executable object code (or data) that is traceable to a requirement and, ~~which~~ by design is either (a) not intended to be executed (code) or used (data), for example, a part of a previously developed software component such as unused legacy code, unused library functions, or future growth code;, or (b) is only executed (code) or used (data) in certain configurations of the target computer environment, for example, code that is enabled by a hardware pin selection or software programmed options. The following examples are often mistakenly categorized as deactivated code but should be identified as required for implementation of the design/requirements: defensive programming structures inserted for robustness, including compiler-inserted object code for range and array index checks, error or exception handling routines, bounds and reasonableness checking, queuing controls, and time stamps.

It is still required that deactivated code be identified during the planning process. Details are now expected on the means to prevent the inadvertently activation of such code. This information should be provided in the plans. The PSAC description is also enhanced to identify deactivated code as part of the “additional considerations”.

Regarding the development processes, DO-178B/ED-12B guidance on deactivated code in the integration process section was moved to an activity description (new section §5.2.4) in the scope of the design process. This becomes consistent with the design data description. This new section highlights the need to design and implement a protection mechanism, and also to develop the deactivated code in the same way that the rest of the code.

Here is the activity description in §5.2.4



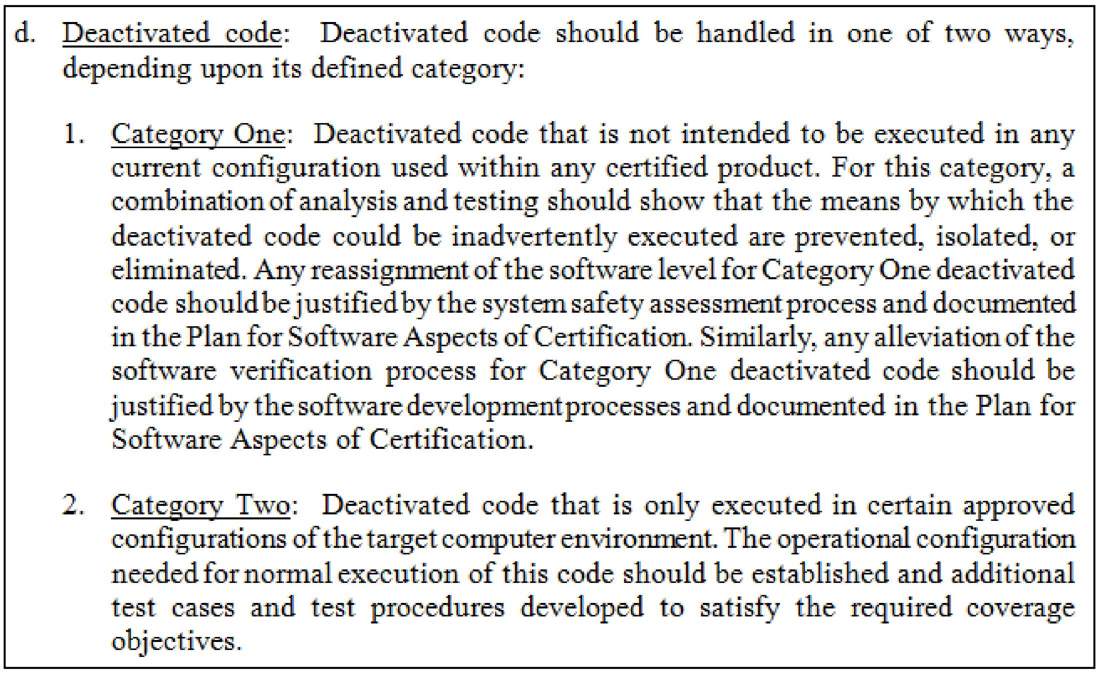
The activities for deactivated code include:

a. A mechanism should be designed and implemented to assure that deactivated functions or components have no adverse impact on active functions or components.

b. Evidence should be available that the deactivated code is disabled for the environments where its use is not intended. Unintended execution of deactivated code due to abnormal system conditions is the same as unintended execution of activated code.

c. The development of deactivated code, like the development of the active code, should comply with the objectives of this document.

In the section about structural coverage analysis resolution, the “two” categories of deactivated code are clarified. Particularly for deactivated code that is not intended to be executed in any configuration, the text opens the door to develop this code at a lower software level, and/or to alleviate the verification activities on this code. §6.4.4.3



d. Deactivated code: Deactivated code should be handled in one of two ways, depending upon its defined category:

1. Category One: Deactivated code that is not intended to be executed in any current configuration used within any certified product. For this category, a combination of analysis and testing should show that the means by which the deactivated code could be inadvertently executed are prevented, isolated, or eliminated. Any reassignment of the software level for Category One deactivated code should be justified by the system safety assessment process and documented in the Plan for Software Aspects of Certification. Similarly, any alleviation of the software verification process for Category One deactivated code should be justified by the software development processes and documented in the Plan for Software Aspects of Certification.

2. Category Two: Deactivated code that is only executed in certain approved 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.

4.3.5 WCET and Stack analysis

WCET (Worst Case Execution Time) and Stack analysis were identified in DO-178B/ED-12B as part of reviews and analysis of the source code verification process (objective “accuracy and consistency” in 6.3.4.f). This could be considered as acceptable 25 years ago, when assembler language was heavily used. But now, it is obvious that time and memory assessment might not be achieved through reviews and analysis of source code.

There were many discussions of this topic, but it was agreed not to significantly change the approach. For example, one proposal was to move these aspects from source code verification to tests, as exercising the executable object code is often necessary to satisfy this objective.

In the end, limited additions were made to try to address this concern.

- In §6.3.4.f, a sentence is added, requiring that compiler, linker and hardware be assessed for impact on WCET.
- In the introduction to the section on software reviews and analysis (§6.3), it is also identified that reviews and analysis alone may not completely satisfy some objectives (e.g. WCET, stack analysis) and that some tests may be also necessary.

§6.3

> There may be cases where the verification objectives described in this section cannot be completely satisfied via reviews and analyses alone. In such cases, those verification objectives may be satisfied with additional testing of the software product. For example, a combination of reviews, analyses, and tests may be developed to establish the worst-case execution time or verification of the stack usage.

FAQ#73 “Are timing measurements during testing sufficient or is a rigorous demonstration of worst-case timing necessary?” was reworked to provide a complete discussion of this topic, but the revision was editorial in nature and doesn’t provide additional information.

4.3.6 Note 6.4.2.1.d on test cases

Two kinds of misunderstandings may occur during the test case development activity. First, the number of test cases to satisfy the objectives for compliance and robustness with HLR/LLR requirements is SW level dependent. Second, the number of test cases is dependent on the structural coverage criteria to be achieved. This second misinterpretation is more critical since it could lead to a structure-based testing approach (e.g. the techniques sometimes referred to as “black box” and “white box”).

While DO-178B/ED-12B enforces the requirement-based test approach, and while the differences from one SW level to another are based on the applicability of the objectives and the independence criteria, but never on the objective itself, the note in §6.4.2.1.d seemed to open the door to the interpretations mentioned above:

> *NOTE: One method is to test all combinations of the variables. For complex expressions, this method is impractical due to the large number of test cases required. A different strategy that ensures the required coverage could be developed. For example, for Level A, the Boolean operators could be verified by analysis or review, and to complement this activity, test cases could be established to provide modified condition/decision coverage.*

The purpose of the note was to provide an example that the satisfaction of the objective may be a combination of review, analysis and tests. But this note introduced a linkage between the number of test cases to be developed and both the SW level and the structural coverage criteria.

To prevent such errors in the future, this note was modified in the form of a FAQ and moved to DO-248C/ED-94C, keeping the initial intent:

FAQ #78: For software requirements expressed by logic equations, how many normal range test cases are necessary to verify the variable usage and the Boolean operators?:

> One method is to test all combinations of the variables. For complex expressions, this method is impractical due to the large number of test cases required. A different strategy could be developed to verify the variable usage and the Boolean operators and to ensure that requirements-based test coverage is achieved. For example, the Boolean operators could be verified by analysis or review. To complement this activity, test cases could be established to demonstrate that each variable correctly influences the output.

4.3.7 Robustness

In DO-178B/ED-12B, robustness testing was sometimes misinterpreted as additional tests that supplemented requirements-based tests. This is clarified now and definitely states that all tests, normal and robustness, should be requirement based.

A note is added in the section §6.4.2 on requirements-based test selection:

> Robustness test cases are requirements-based. The robustness testing criteria cannot be fully satisfied if the software requirements do not specify the correct software response to abnormal conditions and inputs. The test cases may reveal inadequacies in the software requirements, in which case the software requirements should be modified. Conversely, if a complete set of requirements exists that covers all abnormal conditions and inputs, the robustness test cases will follow from those software requirements.

To be more flexible, it is also recognized that some mechanisms as described in the standards may also be used to improve robustness. So, implicitly, some robustness tests should be developed to assess the correctness of the implementation of these mechanisms.

As a result, some additional information is provided in section 4.5 on Software Development Standards:

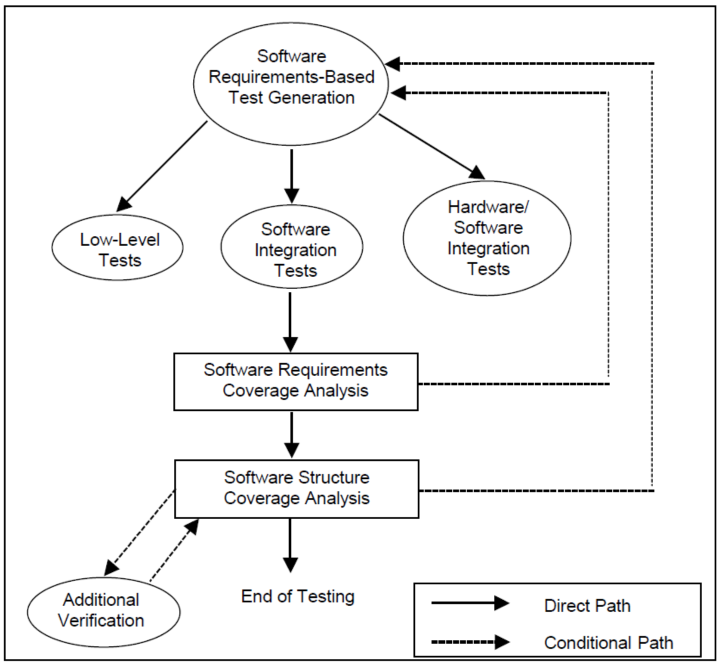
> d. Robustness should be considered in the software development standards.
>
> Note 1: In developing standards, consideration can be given to previous experience. Constraints and rules on development, design, and coding methods can be included to control complexity. Defensive programming practices may be considered to improve robustness.

Additional text is also provided in FAQ#32 in DO-248C/ED-94C (***What are defensive programming practices?***). This FAQ makes a connection between programming practices and robustness but explains that programming practices don't supersede the need for requirements specifying the correct software response to abnormal conditions and inputs.

4.3.8 Figure 6.1 on tests and reverification

§6.4 illustrates the relationships between the three types of tests (HW SW integration testing, SW integration testing, and Low-Level testing) and the requirements coverage analysis first, then the structural coverage analysis. The intent of the figure was to clarify that these two coverage analysis are based on the full set of test cases.

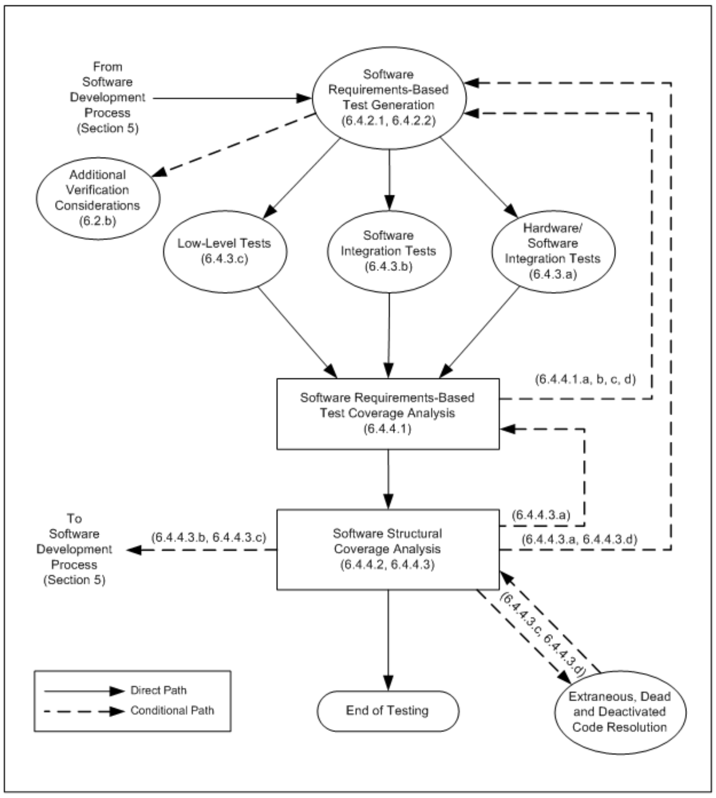
Figure §6.1


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

There was some discussion of the meaning of the "Additional Verification" bubble: Is it the additional verification required if SW level is A and there is not direct traceability between source code and object (§6.4.4.2)? Does this relate to any considerations on the compiler (§4.4.2)?

Whatever the correct answer, this discussion opened the door to a brand new figure, more complete, (e.g. including the structural coverage analysis resolution) and linking all the applicable subsections.

New figure 6.1:


From Software Development Process (Section 5)
Software Requirements-Based Test Generation (6.4.2.1, 6.4.2.2)
Additional Verification Considerations (6.2.b)
Low-Level Tests (6.4.3.c)
Software Integration Tests (6.4.3.b)
Hardware/ Software Integration Tests (6.4.3.a)
(6.4.4.1.a, b, c, d)
Software Requirements-Based Test Coverage Analysis (6.4.4.1)
(6.4.4.3.a)
To Software Development Process (Section 5)
(6.4.4.3.b, 6.4.4.3.c)
Software Structural Coverage Analysis (6.4.4.2, 6.4.4.3)
(6.4.4.3.a, 6.4.4.3.d)
(6.4.4.3.c, 6.4.4.3.d)
Direct Path
Conditional Path
End of Testing
Extraneous, Dead and Deactivated Code Resolution

The additional verification is no longer linked to the "structural coverage analysis" but now refers to section 6.2.b about the "necessary additional analysis to be conducted "*when it is not possible to verify specific software requirements by exercising the software in a realistic test environment*".(Text unchanged from DO-178B/ED-12B)

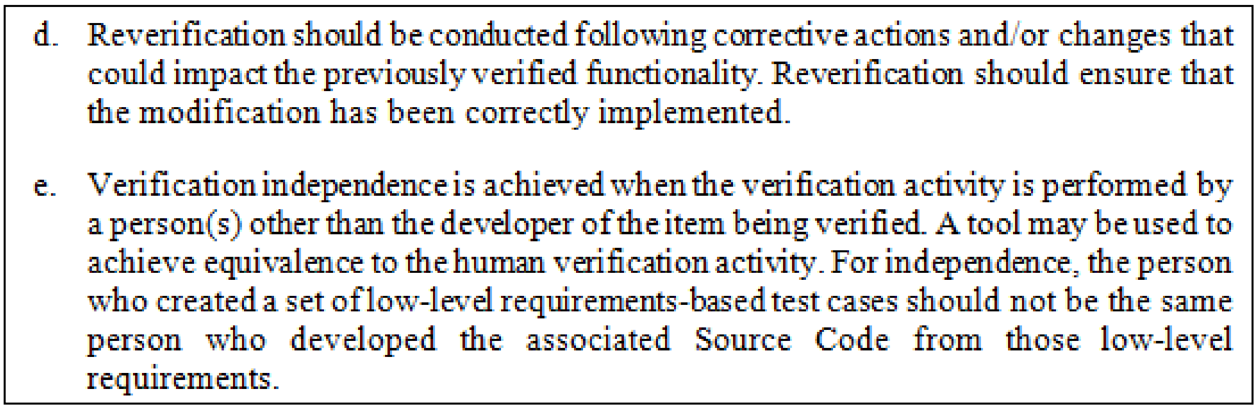
In DO-178B/ED-12B, considerations on "reverification" had to be addressed as part of the Software Verification Plan (§11.3) but no guidance was provided in the verification process. This omission is corrected with the addition of specific considerations in the description of the overview of the verification process, §6.2 (see next section).

4.3.9 Additional considerations in verification process

The section providing an overview of the software verification process activities (6.2) adds considerations on:

- Reverification aspects, which are mainly extracted from FAQ#58 of DO-248/ED-94
- Independence; that is, a new wording of the glossary definition of the term.

These considerations are of course not new, but were not sufficiently identified in the verification process section. This gap is now filled as follows: §6.2



d. Reverification should be conducted following corrective actions and/or changes that could impact the previously verified functionality. Reverification should ensure that the modification has been correctly implemented.

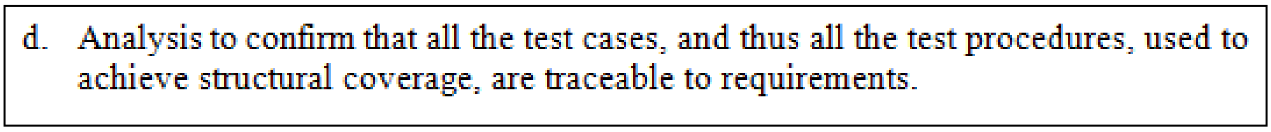
e. Verification independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified. A tool may be used to achieve equivalence to the human verification activity. For independence, the person who created a set of low-level requirements-based test cases should not be the same person who developed the associated Source Code from those low-level requirements.

4.3.10 Structural coverage analysis

Two concerns were identified on this topic: Structural testing (i.e., testing based on the code structure) and data and control coupling.

On the first concern, an additional bullet (d) is added in section 6.4.4.1 (Requirements based test coverage analysis): This bullet provides a hook from "*A Requirements Based Testing"* dedicated section to the "*Structural Coverage Analysis*". It is now clearly explained that only the tests based on requirements are valuable for structural coverage analysis, and an analysis may be necessary for demonstration:

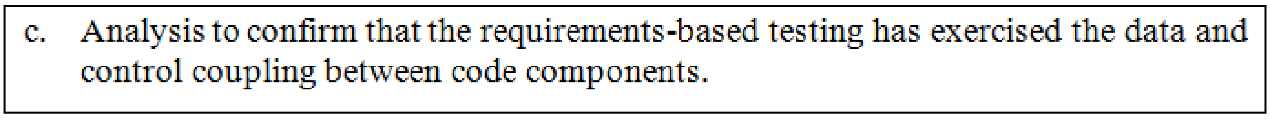
§6.4.4.1



d. Analysis to confirm that all the test cases, and thus all the test procedures, used to achieve structural coverage, are traceable to requirements.

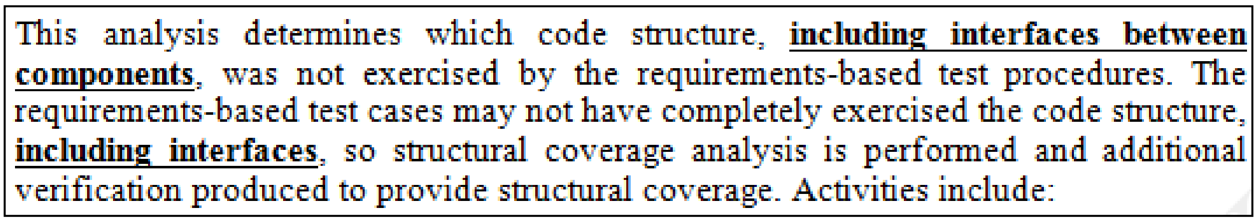
For data and control coupling, it was necessary to re-affirm that objective A7-8 is not a verification of the data/control coupling. Data and Control coupling are defined in the design data as part of the architecture. Verification of this architecture, including interfaces between components, is part of the verification of the outputs of the design data (table A-4). Compliance of source code to this architecture is also verified as part of the verification of the source code (table A-5). Objective A7-8 is related to the structural coverage analysis, and thus to the verification of test data. Therefore, the activity needed to satisfy the objective consists in analyzing how well the requirements-based tests fully exercised the coupling between the components.

§6.4.4.2.c


c. Analysis to confirm that the requirements-based testing has exercised the data and control coupling between code components.

To emphasize the above clarification, the introduction text in the Structural Coverage Analysis section (§6.4.4.2) now lists the “interfaces between the components” as an input of the Analysis.

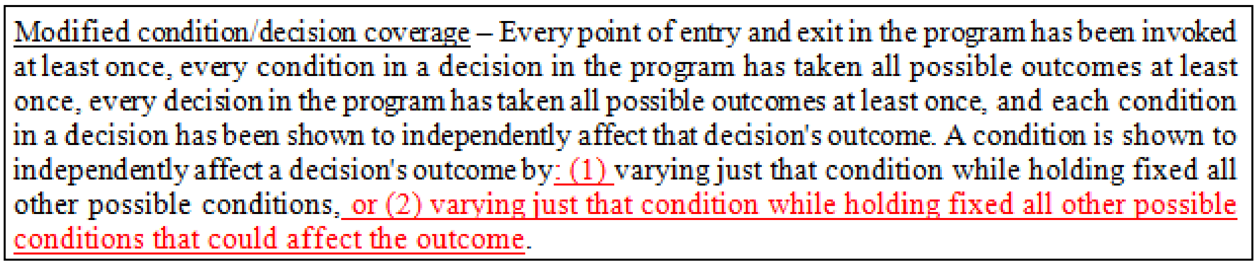
Changes in § 6.4.4.2.


This analysis determines which code structure, **including interfaces between components**, was not exercised by the requirements-based test procedures. The requirements-based test cases may not have completely exercised the code structure, **including interfaces**, so structural coverage analysis is performed and additional verification produced to provide structural coverage. Activities include:

An example of data and control coupling is provided in the modified FAQ#67 of DO-248C/ED-94C. This FAQ also identifies the typical test cases that should be developed to satisfy the 4.4.1.d objective for this example.

4.3.11 MC/DC definition

“Modified Condition/Decision Coverage” (MC/DC) leads to much more discussion on the exact intent, and how to apply it depending on source code language, than on how it helps really to detect errors. In the DO-178C/ED-12C glossary, the definition is slightly changed, to extend the way that a condition may affect independently a decision:


Modified condition/decision coverage – Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by: (1) varying just that condition while holding fixed all other possible conditions, or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome.

This new definition is so “clear”, that it was necessary to update the DP#13 in DO-248C/ED-94C, to understand the meaning of this change.

This DP#13 provides other examples of statement, decision, and MC/DC coverage, but it mainly adds new terms. The intent is to address some possible features of programming languages.

- Unique cause MC/DC (regular)

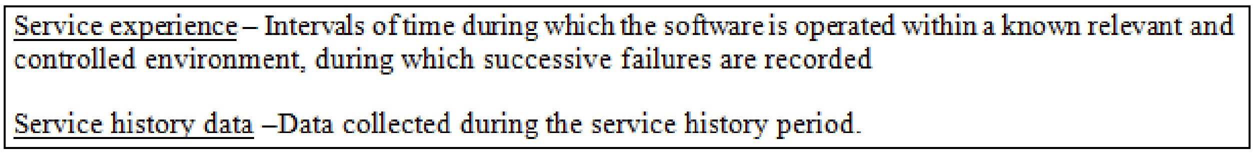
- Masking MC/DC and Short circuit MC/DC (when the complete condition sequence is not evaluated)
- Coupled conditions (same variable used in multiple conditions)

In short, the new definition of MC/DC officially permits the “masking MC/DC” variation that had been approved in practice for DO-178B/ED-12B. This approach allows a wider set of test vectors for MC/DC. It can handle some decisions with coupled conditions that are not amenable to unique cause MC/DC since the coupling prevents varying one condition while keeping the coupled condition fixed.

4.3.12 Service history

Service history is still identified as an alternative method (§12.3.4). Considerations on this approach are enhanced to make the method more applicable.

New definitions appear in the glossary, in addition of "Product service history":



Service experience – Intervals of time during which the software is operated within a known relevant and controlled environment, during which successive failures are recorded

Service history data –Data collected during the service history period.

A new criterion for acceptability is added in the introduction of section §12.3.4: "*Length of the product service history*". Then the previous guidance is replaced with much more comprehensive considerations, related to:

- **The relevance of the service history**: Type of service history (flight hours, for software used continuously or number of demands for software executed on demand e.g; landing gear software) in regard of the type of software, known configuration, operating time collection process (means to collect and calculate the service history data), changes to the software during the product service history, usage and environment to show the relevance of the service history data, deactivated code (if a part of code was not activated during the period of service history)
- **Sufficiency of accumulated service history:** No quantitative guidance, but only considerations that if more credit is claimed by using this alternate approach, then more accumulated service history is required. The credit is assessed through system safety objectives, environments (same or not), software objectives, and the amount of other evidence in addition to service history
- **Collecting, reporting and analyzing problems found during service history:** The required details concerning problems that have occurred during the service history. Information on the problems themselves, and also on process-related problems and on safety-related problems.
- **Detailed information to be included in the PSAC**

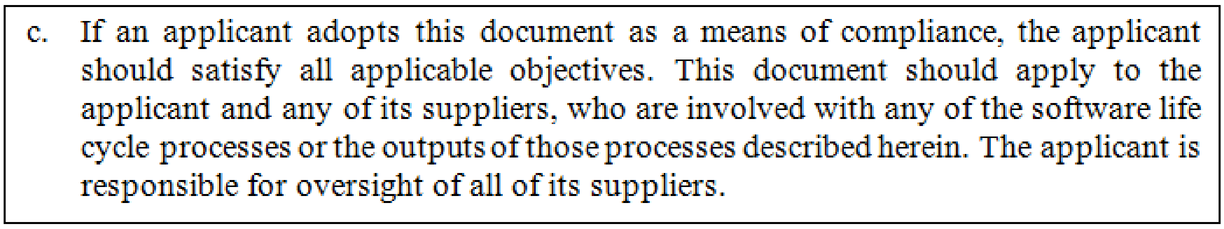
In short, additional guidance is provided on this topic. The possible certification credit claimed by using this "alternative approach" is further explained in DO-248/ED-94C (DP#4).

4.3.13 Supplier monitoring

While DO-178C/ED-12C still does not provide any requirements on the applicant's internal organization, the growing complexity of modern industrial work sharing, and in particular, the typical involvement of subcontracting organizations in the production of software lifecycle data, raised the need to enhance the supplier monitoring aspects in DO-178C/ED-12C.

It is now explicit that the applicant is responsible for the oversight of its suppliers, and that the same certification basis is applicable to them.

§1.4



c. If an applicant adopts this document as a means of compliance, the applicant should satisfy all applicable objectives. This document should apply to the applicant and any of its suppliers, who are involved with any of the software life cycle processes or the outputs of those processes described herein. The applicant is responsible for oversight of all of its suppliers.

DO-178C/ED-12C requires that the means to oversee the suppliers be addressed during the planning process (§4.2), and described in the plans (PSAC, SCMP, SQAP). The purpose of this oversight is to ensure that the supplier processes and outputs comply with approved software plans and standards. This oversight activity is now added in the SCM process (§7.2) and SQA process (§8.2 bullet i). In addition the SAS should also provide a status on this compliance.

How does the contribution of a supplier impact the set of approved plans and standards? DO-178C/ED-12C does not explicitly require that the supplier plans and standards be included in the set of approved documents. However, if the supplier applies its own processes, and if these processes are described in a set of plans and standards, it seems more than logical to review and approve these documents. On top of that, the supplier activities are sharply linked to the main applicant processes, and the compatibility of the supplier methods with the applicant methods may also need to be assessed. These aspects should be considered during the review of the Software Planning Process.

4.4 Hidden objectives

A few objectives are now clearly identified in DO-178C/ED-12C while they already existed in DO-178B/ED-12B but weren't identified in the objectives tables or in the text.

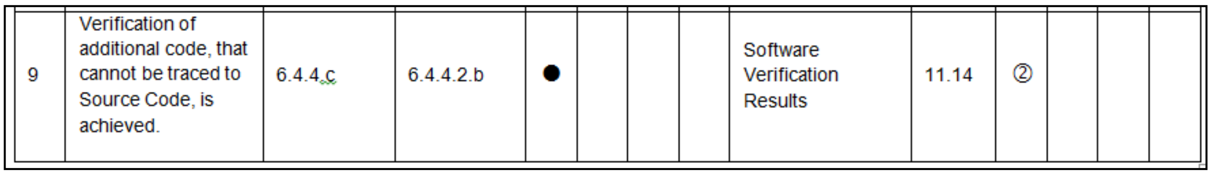
4.4.1 Verification of additional code

Objectives of the "Test Coverage Analysis" are now clearly listed in section 6.4.4, for both requirements coverage and structural coverage. Regarding structural coverage, the code coverage objective is based on the software level. The text doesn't provide the explicit coverage criteria that are applicable for each software level but uses the words "applicable coverage criteria" or "appropriate to the software level". To know more about these criteria, one needs to refer to the objective tables where the words "statement", "decision" and "MC/DC" are used. On this point, DO-178C/ED-12C did not bring in much added value.

In addition, under certain conditions DO-178B/ED-12B section 6.4.4.2.b required the applicant to verify the traceability between Source Code and Object Code. In case of non-direct traceability, additional verification activities were expected in order to demonstrate the correctness of the generated code sequences. There was nothing about this in the A-7 objective table.

This DO-178B/ED-12B text and the associated expectations have often been misunderstood. CAST papers addressed this topic. Therefore, clarifications were needed and are now incorporated in DO-178C/ED-12C:

- In DO-178B/ED-12B, the wording "*the analysis may be performed on the source code, unless …*" seemed to suggest that the adequate level was the object code, which was not the initial intent. The text has been updated: Structural coverage analysis may now be performed at any level, i.e, source code, object code or executable object code. It is up to the applicant to choose the most appropriate level.
- Independent of the form of the code used to perform the structural coverage analysis, if the software level is A, an analysis of the code produced by compiler/linker or other tools used to generate the executable object code needs to be conducted. If such tools generate additional code sequences that are not directly traceable to source code, additional verification should be conducted.
- The meaning of the words "direct traceability" is now clarified in a note. It explains that "branches" and "side effects" should be considered.
- The table A-7 reflects this change. A new objective, applicable only to level A is added:



9	Verification of additional code, that cannot be traced to Source Code, is achieved.	6.4.4.c	6.4.4.2.b	●				Software Verification Results	11.14	②			

It should be noted that compared to DO-178B/ED-12B, there is no extra information in DO-178C/ED-12C regarding the exact nature of this "additional verification". The guidance remains focused on the general verification objective consisting in establishing *“the correctness of such generated code sequences*”.

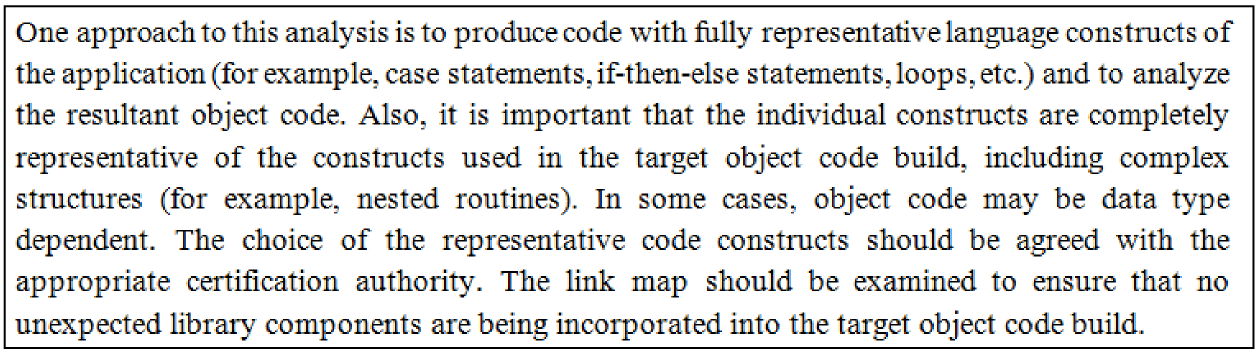
Regarding this additional verification, DO-178B/ED-12B and CAST paper #12 were clearly limited to the compiler effects. The new text in section 6.4.4.2.b brings the other generation tools in the game. Therefore, the activity may no longer be limited to traceability analysis between source code and object code but may also need to consider the effects of all tools used in the Executable Object Code Generation chain.

Based on this wider scope, it is no longer acceptable, as stated in the CAST position paper #12 to limit the activities to the analysis between source code and assembly listings. As a minimum, an additional analysis is needed to assess the impact of the other generation tools on the execution paths within the executable object code.

Some interesting information on the “verification of additional code” topic is also provided in DO-248C/ED-94C:

- FAQ#42: An additional question is included in this FAQ “*Is there any consideration for Level A software when Object Code Coverage (OCC) analysis is performed and when the compiler generates Object Code that is not directly traceable to Source Code statements?”* The answer to this question is “yes”. The FAQ explains that the plans and standards should address different items that may impact the OCC analysis, such as coding restrictions and limitations, compiler restrictions and limitations, complexity limitations, etc. Then it proposes to include in the OCC analysis the following topic:
    - *Data that substantiates that the compiler produces the Object Code expected as assumed in the OCC approach*
    - *Data that substantiates that the results achieved from the OCC method provide the appropriate coverage assurance.*
    - *Data showing how each of the following (and any other relevant OCC topics) are addressed:* and it lists several features such as jump statements, bitwise operators, optimizations, to be considered.

- DP#12: This DP deals with “**Object code to Source code traceability issues**”; it remains unchanged and focused on the traceability approach. The key point of this DP is to propose to base the traceability analysis on a set of representative source code sequences, just like CAST paper #12 does.


One approach to this analysis is to produce code with fully representative language constructs of the application (for example, case statements, if-then-else statements, loops, etc.) and to analyze the resultant object code. Also, it is important that the individual constructs are completely representative of the constructs used in the target object code build, including complex structures (for example, nested routines). In some cases, object code may be data type dependent. The choice of the representative code constructs should be agreed with the appropriate certification authority. The link map should be examined to ensure that no unexpected library components are being incorporated into the target object code build.

It is interesting to notice that no information was added in DO-248C/ED-94C on the core change requiring an analysis of the code produced by compiler/linker or other tools used to generate the executable object. As for rev B, DO-248C/ED-94C is only focused on compiler effects.

4.4.2 SQA objectives

In DO-178B/ED-12B all the objectives applicable to the SQA process were not clearly identified in section 8. Some section 4 objectives on Software planning process (table A-1 objectives 6 and 7 “Software Plans comply with this document” and “software plans are coordinated”) need to be also achieved by SQA. Evidence of that is that the output data identified in the table are both Software verification results and SQA records. These two objectives refer to section 4.6 review and assurance of the software planning process.

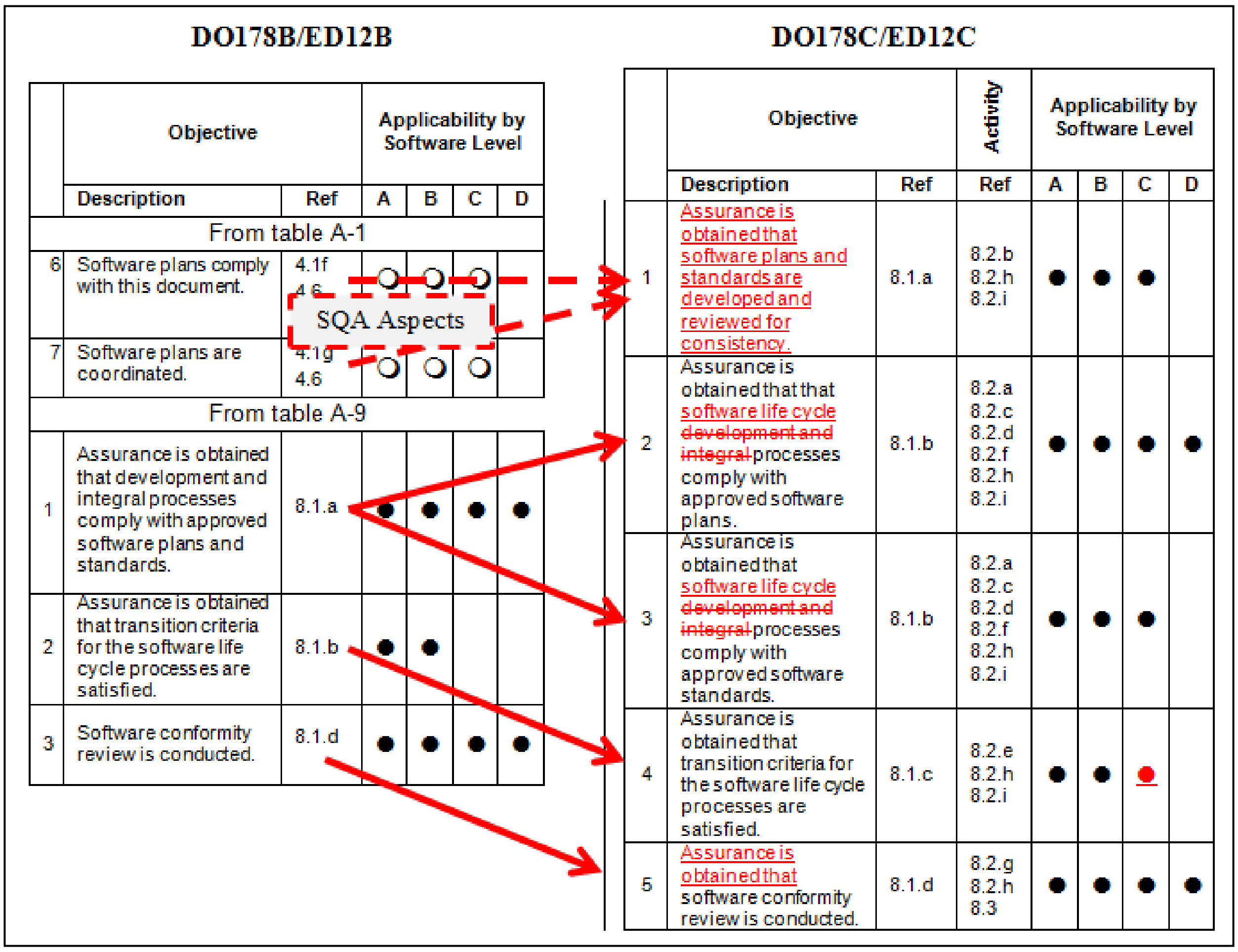
In DO-178C/ED-12C, changes were performed to merge all SQA objectives into the same section and into the same objective tables. This approach raised the need to clearly identify some hidden objectives of the SQA process.

First in section 4.6, the term “assurance” was removed, and table A-1 modified to suppress the SQA records as an output of the two objectives 6 and 7. The wording of the objective 7 was reworded to clarify its intent in “Development and revision of Software Plans are coordinated”. But the two objectives still identified SQA records as an output.

However, the SQA role in the oversight of the planning process is identified in section 8.1 in form of a new objective: “*Obtain assurance that software plans and standards are developed and reviewed for consistency*”

Then the next objective “*Obtain assurance that the software life cycle processes comply with approved plans and standards*” may be correctly applied. It should be noted that this objective is split in two in the objective table, to clarify the leveling. For level D, the objective related to the standard is not applicable.

Here is a comparative view of the changes:



SQA Aspects

DO178B/ED12B

	Objective		Applicability by Software Level			
	Description	Ref	A	B	C	D
From table A-1						
6	Software plans comply with this document.	4.1f 4.6	○	○	○	
7	Software plans are coordinated.	4.1g 4.6	○	○	○	
From table A-9						
1	Assurance is obtained that development and integral processes comply with approved software plans and standards.	8.1.a	●	●	●	●
2	Assurance is obtained that transition criteria for the software life cycle processes are satisfied.	8.1.b	●	●		
3	Software conformity review is conducted.	8.1.d	●	●	●	●

DO178C/ED12C

	Objective		Activity	Applicability by Software Level			
	Description	Ref	Ref	A	B	C	D
1	Assurance is obtained that software plans and standards are developed and reviewed for consistency.	8.1.a	8.2.b 8.2.h 8.2.i	●	●	●	
2	Assurance is obtained that that software life cycle ~~development and integral~~ processes comply with approved software plans.	8.1.b	8.2.a 8.2.c 8.2.d 8.2.f 8.2.h 8.2.i	●	●	●	●
3	Assurance is obtained that software life cycle ~~development and integral~~ processes comply with approved software standards.	8.1.b	8.2.a 8.2.c 8.2.d 8.2.f 8.2.h 8.2.i	●	●	●	
4	Assurance is obtained that transition criteria for the software life cycle processes are satisfied.	8.1.c	8.2.e 8.2.h 8.2.i	●	●	●	
5	Assurance is obtained that software conformity review is conducted.	8.1.d	8.2.g 8.2.h 8.3	●	●	●	●

Beyond the editorial changes in the objectives wording, the main impact is the application of objective 4 (transition criteria) to the level C software.

In addition, it should be noted that the last objective “Software conformity review is conducted” is rewritten to be consistent with the text of 8.1. The guidance of the document doesn’t require that the software conformity review should be conducted by the SQA responsible that is often understood. Guidance only requires that the SQA “obtain assurance” that the software conformity review. The objective is now in line with the text.

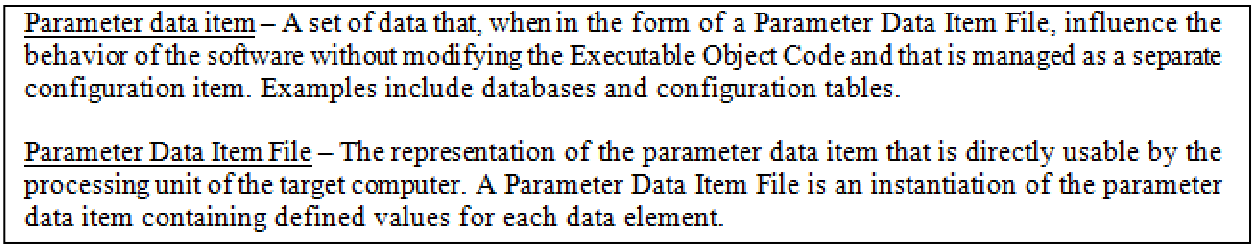
4.5 Gaps and New Topics

DO-178C/ED-12C addressed several specific issues that resulted in changes to only one or two paragraphs. Each such change may have an impact upon the applicant as these changes either addressed clear gaps in DO-178B or clarified guidance that was subject to differing interpretations.

4.5.1 Parameter data item (PDI)

This new topic is intended to address the possibility to produce and modify some configuration tables or databases separately from the executable object code. The guidance is applicable when such data is modified and the executable object code is not reverified. Previously this issue was addressed through some CRI, particularly in IMA context.

The new text identifies two keywords “parameter data item” (general word) and “parameter data item file” (the executable representation of the PDI):



Parameter data item – A set of data that, when in the form of a Parameter Data Item File, influence the behavior of the software without modifying the Executable Object Code and that is managed as a separate configuration item. Examples include databases and configuration tables.

Parameter Data Item File – The representation of the parameter data item that is directly usable by the processing unit of the target computer. A Parameter Data Item File is an instantiation of the parameter data item containing defined values for each data element.

As the PDI file is separate from the executable object code, throughout the document multiple changes were performed to replace “*executable object code*” with “*executable object code and Parameter Date Item Files*”, as all considerations on generation, identification and management of the executable object code are applicable to the Parameter data Item Files.

The use of a PDI impacts all the processes:

- A new section is added in the “Software Considerations in the System Life Cycle Processes” to highlight the possible impact on this approach on the system
- During the planning process, processes applicable to PDI should be defined and described in the plans, in particularly in the PSAC, as additional considerations. The software load control and compatibility aspects should be also addressed
- PDI is subject to High Level Requirements development. These requirements define the structure, attributes and (when applicable) the values. This is often called “usage domain”. The choice to consider these data as HLR and not LLR is to make the guidance applicable to level D software.
- In the integration process, the PDI files are generated
- Of key importance is the new section 6.6 on the verification of the PDI. This section defines under which conditions the verification of the PDI may be conducted separately from the executable object code. These conditions are tied to the coverage of the executable object code verification Demonstration that the executable object code is able to handle the PDI values inside the limits provided by the PDI HLR, and to be robust against invalid structures and/or attributes, needs to be provided.
- The verification objectives on the PDI file itself, conducted separately, are defined, and summarized in the table A-5. The first objective is to verify that the PDI file is

compliant with its HLR (structure, attributes), and that it doesn't contain any unintended element. This objective includes also the verification of the correctness and consistency of the element values (not only that it is in the range defined in the HLR). The second objective is to verify the completeness of the verification.

Table A-5 extract

8	Parameter Data Item File is correct and complete	6.6.a	6.6	●	●	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
9	Verification of Parameter Data Item File is achieved.	6.6.b	6.6	●	●	○		Software Verification Results	11.14	②	②	②	

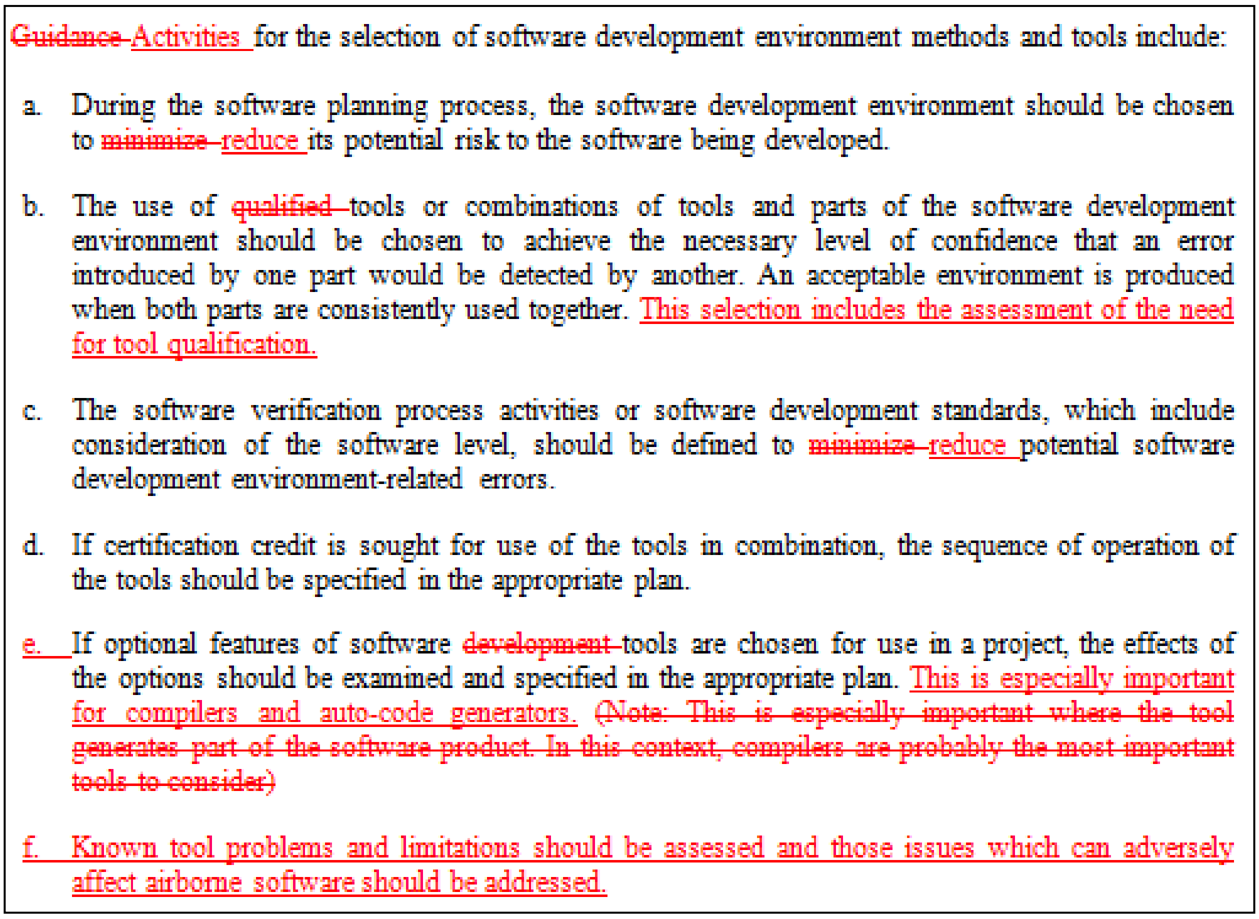
It should be noted that the PDI file is identified as Software Life Cycle Data (§11.22), and is also the topic of a discussion paper (DP#20) in DO-248C/ED-94C, providing clarifications and examples.

- What is a parameter data item?
- What considerations apply with respect to compatibility between Executable Object Code and Parameter Data Item Files?
- What does the fourth bullet of section DO-178C/DO-278A section 6.6 mean?
- How is a separately verifiable Parameter Data Item File verified?
- How is PDI related to option-selectable software, user-modifiable software, and field-loadable software?
- What is the purpose of the "normal range testing" and the "robustness of the EOC" defined in DO-178C/DO-278A section 6.6?
- What is the rationale for assigning the PDI the same software/assurance level as the component using it and not using the assignment rules in DO-178C/DO-278A section 2.3.4?

4.5.2 Software Development Environment

The DO-178B/ED-12B "**guidance**" for selection of methods and tools used to develop the software becomes in DO-178C/ED-12C some **activities.** And these activities are referenced in table A-1 Software planning process in the context of objective 4 which is enhanced as "Software Life Cycle Environment is **selected** and defined".

Beyond these clarifications, it should be noted that several changes modify the scope of this section. Here the new section §4.4.1 compared to DO-178B/ED-12B:



~~Guidance~~ Activities for the selection of software development environment methods and tools include:

a. During the software planning process, the software development environment should be chosen to ~~minimize~~ reduce its potential risk to the software being developed.

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. This selection includes the assessment of the need for tool qualification.

c. The software verification process activities or software development standards, which include consideration of the software level, should be defined to ~~minimize~~ reduce 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. This is especially important for compilers and auto-code generators. ~~(Note: This is especially important where the tool generates part of the software product. In this context, compilers are probably the most important tools to consider)~~

f. Known tool problems and limitations should be assessed and those issues which can adversely affect airborne software should be addressed.

- Bullet b about the selection of development tools is no longer limited to the "qualified" tools. Therefore the assessment of the need for tool qualification becomes a part of this activity.
- The removal of the word "development" in bullet "e" may be interpreted as an extension to the activity to all the software tools and not only tools used in the framework of the development processes. But as, this section addresses the Software **Development** Environment, and not the Software **Life Cycle** Environment, this change has no impact.
- The note after bullet "e" was included in the text, but also modified. It emphases the need to consider "especially" not only the compiler, but also the auto-code generators. So, whether the auto-code generator is qualified or not, the "*effects of the options should be examined and specified in the appropriate plans*". On the one hand, the development processes should be documented including the way to use the tools. But on the other hand, the identification of options used for a non-qualified auto-code generator may be considered as useless, as the outputs are verified, In some cases, this consideration may be important, as for example, when the outputs of the non-qualified auto-code generator is verified by other tools.
- An additional bullet is included to address the "known tool problems and limitations". This is new, but could be considered as a "good practice". As the development process may rely on multiple COTS tools, the tool selection needs to include information from the vendor on the status of the tool, in terms of known

errors. Then an assessment of these problems in the operation/user context should be conducted and tool limitations defined. As clarified by the new FAQ#83 of DO-248C/ED-94C “*Should compiler errata be consider?*” this activity is applicable to the compiler.

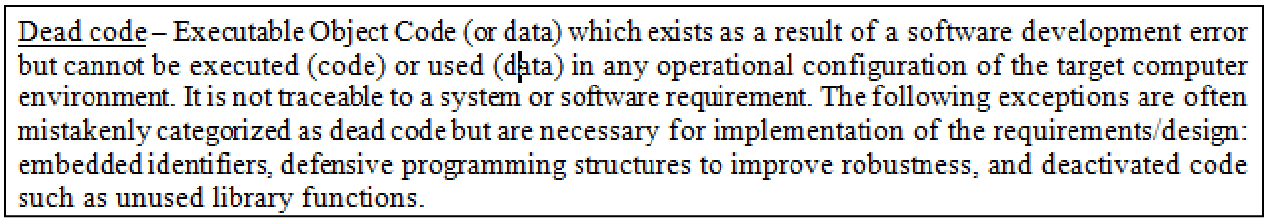
- In section 5.3.2 (coding activities) concerns were raised on the use of autocode generators. A new bullet was added to identify that the use of an autocode generator should conform to the planning process. This is applicable whatever the autocode generator is qualified or not. One may argue that in the case of an autocode generator is not qualified, since the outputs are verified no constraints should be raised on the use of the autocode generator itself. But the community felt it important that, for such critical tools, their usage should be clearly defined during the planning process, and then the process using these tools should be conducted in compliance with the intended use defined in the plans.

4.5.3 Extraneous code

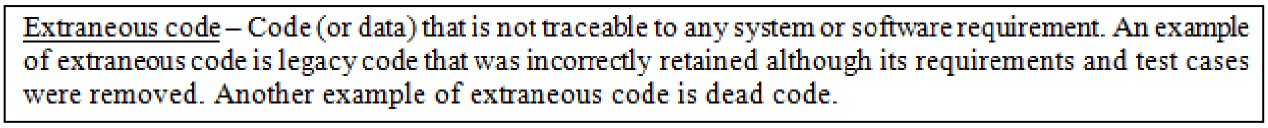
A new term appears in the structural coverage analysis resolution section (§6.4.4.3): Extraneous code, which is an extension of "dead code". The idea is to consider all code (or data) that is the result on an error, whatever this code may be or not exercised. The definition of dead code is limited to the executable object code that cannot be exercised. Extraneous code includes dead code, but also all pieces of code, found at source or object code level, that may be exercised or not.

The fundamental idea remains unchanged. This code, executable or not, should be removed. However, it is now allowed to keep this code as long as it is demonstrated that (1) this extraneous code does not exist in the executable object code, and (2) procedures exist to prevent their inclusion in future software releases.

Here is the definition of "extraneous code" and the new definition of "dead code" providing more examples on exceptions:



Dead code – Executable Object Code (or data) which exists as a result of a software development error but cannot be executed (code) or used (data) in any operational configuration of the target computer environment. It is not traceable to a system or software requirement. The following exceptions are often mistakenly categorized as dead code but are necessary for implementation of the requirements/design: embedded identifiers, defensive programming structures to improve robustness, and deactivated code such as unused library functions.



Extraneous code – Code (or data) that is not traceable to any system or software requirement. An example of extraneous code is legacy code that was incorrectly retained although its requirements and test cases were removed. Another example of extraneous code is dead code.

4.5.4 New Tool qualification criteria

The need for tool qualification and the guidance to qualify the tools were provided in section 12.2 of DO-178B/ED-12B. Two tool qualification criteria were identified

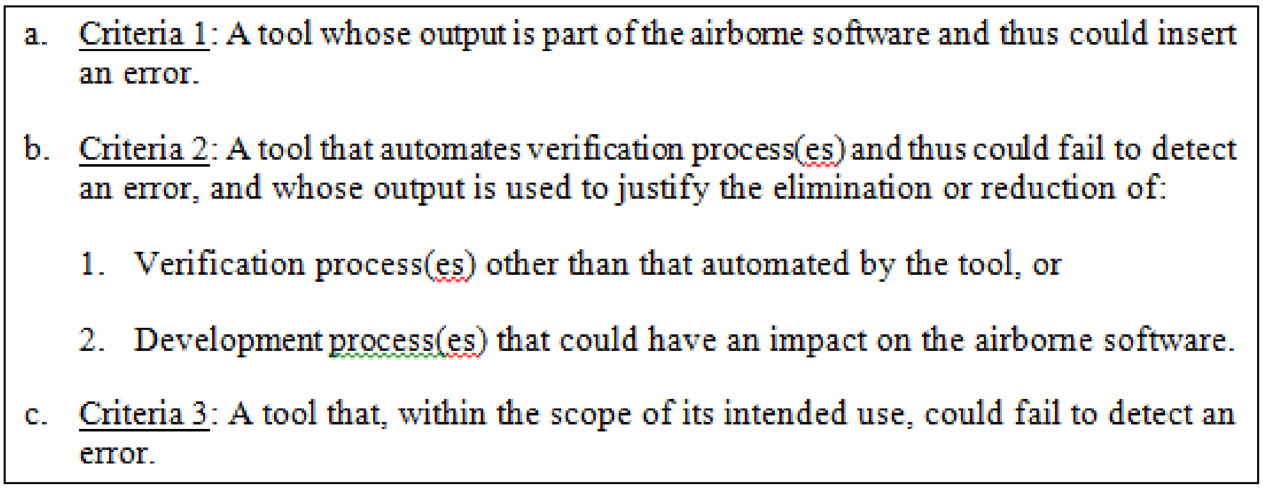
- Tools that may inject an error in the resulting software without its outputs verified were classified as "development tool". For these tools the qualification guidance was to apply the same objectives as for the resulting software.
- Tools that may fail to detect an error were classified as "verification tools". For these tools, the qualification criteria were to demonstrate that the tool complies with its operational requirements under normal operational conditions.

DO-178B/ED-12B's approach to tool qualification raised several issues. First, the terms used to define the tool categories ("development" and "verification") might have imputed functionality to the tools that might have been incorrect. Second, the application of airborne software objectives to qualify development tools was not always relevant (for example, stack overflow during tool execution does not have the same failure consequences as stack overflow during the execution of airborne software) and gave rise to multiple interpretations.

Section 12.2 was rewritten in DO-178C/ED-12C, and the terms "development tool" and "verification tool" have been replaced by three tool qualification criteria that determine the applicable tool qualification level (TQL) depending on the software level. The guidance to qualify a tool is absent from DO-178C/ED-12C, but is provided in the separate DO-330/ED-215 document, domain independent, referenced in section 12.2.

"Criterion 1" addresses the former "development tools", while the two other criteria split the former "verification tools" depending of the certification credit claimed by the qualification of the tool.

Here are the three criteria: §12.2.2



a. Criteria 1: A tool whose output is part of the airborne software and thus could insert an error.

b. Criteria 2: A tool that automates verification process(es) and thus could fail to detect an error, and whose output is used to justify the elimination or reduction of:

   1. Verification process(es) other than that automated by the tool, or

   2. Development process(es) that could have an impact on the airborne software.

c. Criteria 3: A tool that, within the scope of its intended use, could fail to detect an error.

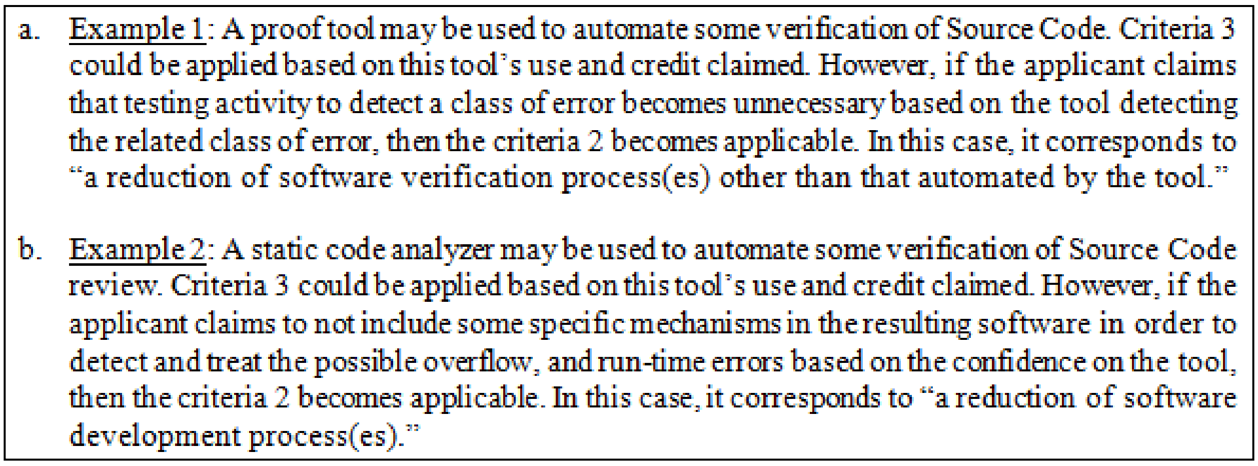
Criterion 3 is the "classic" use of a verification tool: The purpose of the tool is to produce or verify an artifact, and the certification credit claim is only on objectives applicable to this artifact.

Examples:

- A tool that produces the tool procedures from the test cases, the certification credit is limited to the correctness of the test procedures (Objectives A7-1).
- The certification credit for a code checker, that verifies the compliance of source code to the coding standard, is limited to the objectives A5-4 Source code is compliant to standard)

The certification credit claimed is extended in case of application of criterion 2 to objectives that are beyond the data directly verified by the tool.

In an appendix of the Tool Qualification Document, a Discussion Paper (DP#5) provides additional rationale about the need for these 3 criteria. It also includes some examples of distinguishing between criteria 2 and 3, using a "proof tool" and a "static code analyzer"

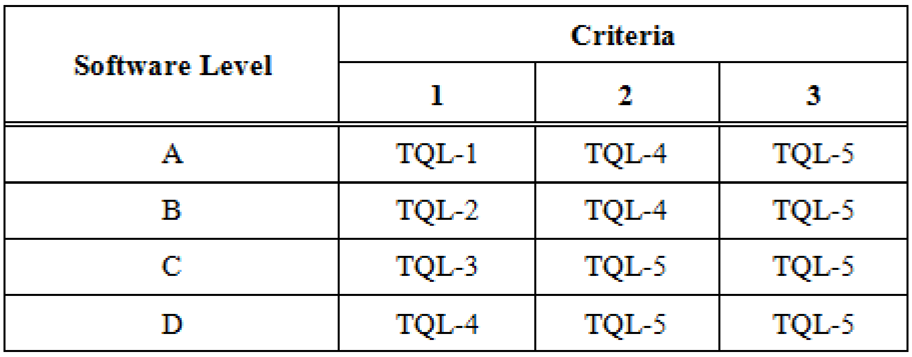


a. Example 1: A proof tool may be used to automate some verification of Source Code. Criteria 3 could be applied based on this tool's use and credit claimed. However, if the applicant claims that testing activity to detect a class of error becomes unnecessary based on the tool detecting the related class of error, then the criteria 2 becomes applicable. In this case, it corresponds to "a reduction of software verification process(es) other than that automated by the tool."

b. Example 2: A static code analyzer may be used to automate some verification of Source Code review. Criteria 3 could be applied based on this tool's use and credit claimed. However, if the applicant claims to not include some specific mechanisms in the resulting software in order to detect and treat the possible overflow, and run-time errors based on the confidence on the tool, then the criteria 2 becomes applicable. In this case, it corresponds to "a reduction of software development process(es)."

The idea is that the software verification process relies on multiple filters to improve the error detection. The certification credit claimed in application of criterion 3 is equivalent to removing one filter, since it has been replaced by the higher level of reliability of the tool. That's why, for these tools, the Tool Qualification Level (TQL) is higher than for a "classic" verification tool,

The applicable TQL is defined in the table 12-1, based on the qualification criteria and on the software level:



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

The TQL applicable for criterion 1 is the replacement for the development tool for each software level, while the TQL-5 for criterion 3 is the replacement for the verification tool in DO-178B/DO-278.

The TQL applicable for Criterion 2 basically requires an increased level of rigor for tools used on software level A and B in order to increase the confidence in the use of the tool (that is, TQL-4 instead of TQL-5). TQL-4 requires that the Tool Requirements data describe all functionality implemented in the tool and provide additional detail about the tool architecture. TQL-4 also requires verification of the compliance of the tool with Tool Requirements. TQL-4 objectives are considered as a minimum to claim confidence in the use of the tool. But the purpose of applying TQL-4 for software level A or B (AL1 and AL2 for DO-278A/ED-109A users) is not to prevent the use of this kind of tool. The following approaches may be considered for tool use:

- In case of deficiencies in the tool life cycle data needed to qualify the tool at TQL-4, the applicant may still use the tool and qualify it at TQL-5; however, other certification/approval credit is limited to the verification objectives of the data under verification.

- In case of COTS, if the data life cycle is not provided by the tool supplier to qualify the tool at level TQL-4, section 11 of this document allows an applicant to augment the data in order to satisfy the objectives for the applicable TQL.

The Tool Qualification Document provides some additional information to explain the rationale to not use the terms “development tools” and “’verification tools” FAQ D1. Another FAQ (FAQ D.5) provides rationale for defining a third tool qualification criterion and includes some examples to help the determination of applicable criteria.

4.5.5 Objectives for Level D software

An inconsistency was identified in the objectives applicable to level D software in DO-178B/ED-12B: Though Table A2 was requiring both Design data and Source Code to be developed; none of the verification objectives associated with these data were applicable (except verification of partitioning integrity).

Let’s consider the Source Code first: In any classical approach of software development, the development of Source Code, regardless of the certification constraints, is by definition needed to provide some inputs to the compiler. Therefore, the applicability of the objective A2-6 (source code is developed) can be said to be a “non-issue” topic, the Source Code being systematically developed. This is true for in-house Software, but when considering the use of COTS (or even PDS), the applicability or non-applicability of this objective makes a significant difference. If Source Code is not required for certification, then the integration of any COTS for which the Source Code is not provided by the COTS vendor is allowed. The same analysis can be extended to design data, which are often not provided by COTS vendors.

In DO-178C/ED-12C, the objectives of the development processes A2-4 (LLR), 5 (Derived LLR) and 6 (Source code) are no longer applicable to level D. The main benefit is to ease the use of COTS and PDS for level D software, as these data do not need to be provided for certification.

A change was consequently added in DO-248C/ED-94C (DP#14) to highlight that to satisfy the partitioning integrity objective, “*even though low-level requirements and Source Code are not required to be submitted as certification evidence, it is necessary to document all details defining the partitioning mechanism.”*

Meanwhile, another inconsistency came up, as all the configuration management objectives are still applicable to level D software, including archival and retrieval: *“Archival and retrieval ensures that the software life cycle data associated with the software product can be retrieved in case of a need to duplicate, regenerate, retest or modify the software product*.” It seems difficult to regenerate the software without the source code!

A new FAQ was thus created to explain that to satisfy some objectives (in particular, the configuration management ones), Low Level Requirements and/or Source Code are necessary, even for level D. The absence of a dot in table A-2 for level D means that evidence of LLR and Source Code is not required, but not that these artifacts are not produced! Undoubtedly further clarification will be necessary on this topic!

4.5.6 Single Event Upsets

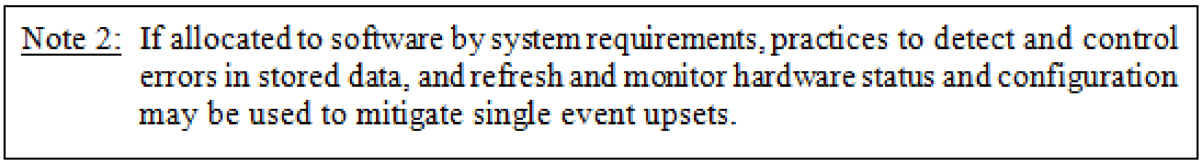
The definition of Single Event Upsets (SEU) is added to the Glossary as “*random bit flips in data that can occur in hardware*.”

Even if the origin of SEU susceptibility is hardware, mitigation for its effects may be made in software or hardware. This could be a decision of a system process, and then that will flow down to the software processes.

In this case, the means to mitigate the SEUs should be provided:

- In the PSAC, in the software overview description.
- In the Software Development Standards:

A note was added in the section §4.5:



Note 2: If allocated to software by system requirements, practices to detect and control errors in stored data, and refresh and monitor hardware status and configuration may be used to mitigate single event upsets.

Additional information is provided on this topic in DO-248C/ED-94C in form of a new DP (DP#21) examining several questions regarding the relationship of single event upset (SEU) to software:

- *What is a Single Event Upset (SEU)?*

  This section supplements the definition provided in the glossary

- *When is it necessary to apply SEU mitigation measures in software?*

  The section recognizes that the origin of SEU susceptibility is hardware and then it would be best if protection could be afforded in hardware. However mitigation for its effects may be made in software or hardware. SEU susceptibility should be identified at system level taking into consideration the type of devices used, the probability of exposure to radiation/bit flip events, and the criticality level of the function.

- *What are some of the SEU mitigation techniques that can be implemented in software?*

  This section provides some of examples of software solutions to the single event problems.

- *What other options are available for SEU mitigation?*

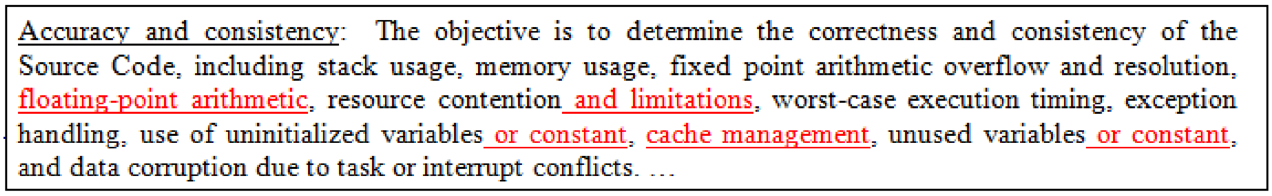
  Some considerations on hardware and architecture to reduce susceptibility to SEUs are provided.

- *What other information is available on single event upset (SEU) mitigation for avionics?*

  This section provides references to Section 3.1.20 of IEC TS 62239 "Process management for avionics – Preparation of an electronic components management plan"

4.5.7 New topics in accuracy and consistency of source code

Section 6.3.4.f provides the objective of "accuracy and consistency of source code". This definition is based on a list of specific items to be considered. The previous list was extended with several new subjects:



Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, memory usage, fixed point arithmetic overflow and resolution, floating-point arithmetic, resource contention and limitations, worst-case execution timing, exception handling, use of uninitialized variables or constant, cache management, unused variables or constant, and data corruption due to task or interrupt conflicts. …

These new topics are only identified in this section and are not further described. However, for two of them, a new DP was added in DO-248C/ED-94C:

- ***DP #16: Cache Management***

The intent of the DP is to handle the incorrect usage or failure of cache memory management that may lead to vulnerabilities which finally could jeopardize the correct software execution. This paper identifies examples of vulnerabilities linked to use of cache memory and identifies some approaches for dealing with those vulnerabilities.

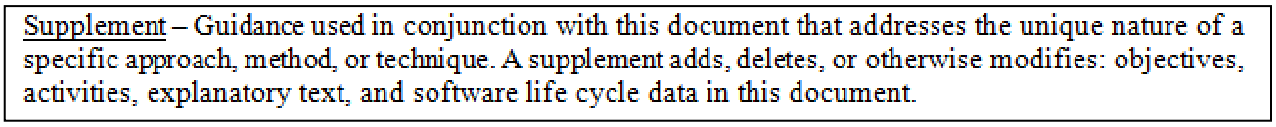
- ***DP #17: Usage of Floating-Point Arithmetic***

The intent of this DP is to address the floating-point arithmetic and numbers usage by defining some considerations which should be addressed. The goal is to ensure that specific vulnerabilities have been properly addressed to ensure the safe usage of floating arithmetic and numbers. The DP provides a list of recommended design and coding constraints.

5. Five supplements and a new document

5.1 Supplement concept

Almost everything is established in the glossary:



Supplement – Guidance used in conjunction with this document that addresses the unique nature of a specific approach, method, or technique. A supplement adds, deletes, or otherwise modifies: objectives, activities, explanatory text, and software life cycle data in this document.

The fundamental idea is to keep the core document as much as possible independent of any methods or techniques, but to extend the guidance by defining more appropriate objectives and/or activities to allow the use of a specific method or technique. A long term objective could be to add in the future new supplements without modifying the core document DO-178C/ED-12C.

Each supplement provides first all information about the method and techniques which are being addressed. Then from section 2 to 12, and annex A (Objectives tables) the supplement uses exactly the same sections and subsections as the core document and identifies:

- If the core document applies: then the text is not duplicated. Only mention "Section X.X of DO-178C or ED-12C is unchanged"
- Or replaces the core document text with a new one, adapted to the method/technique addressed by the supplement
- If new subsections are necessary, they are added at the end of the related section.

Section numbering in the supplements are prefixed with a unique identifier (e.g "MB" for Model based).

Each supplement also contains an appendix for additional information related to the method/techniques addressed. Some FAQ and DP are provided to clarify the technical content of the supplement. Normally this information would be in DO-248C/ED-94C, but to ease the use of the supplement, it was decided to keep this information inside the supplement, and thus in an appendix.

5.2 How to use a supplement

A supplement should be considered at the same level as the core document. More, if a supplement exists for a specific method or techniques, it should be used.

Three supplements were developed in the scope of SC-205/WG-71:

- Formal methods
- Object Oriented Technologies and related techniques
- Model Based development and verification

It is the responsibility of the applicant to ensure that the supplement's use is acceptable to the appropriate certification authority. Supplements are used in conjunction with this document and may be used in conjunction with one another.

As a required part of the software planning process, the applicant should review all potentially relevant supplements and identify those that will be used.

If one or several supplements is considered as applicable, then the plans, and specifically the PSAC, should identify the impact of the use of the selected supplement on the objectives to be satisfied. Then the planning process objective "*Software plans that comply with sections 4.3 and 11 have been produced*" (§4.1.f) will be satisfied when the software plans are compliant with the text of sections 4.3 and 11 as adapted by the supplement (where applicable).

5.3 Tool Qualification Document

SC-205/WG-71 considered that it was necessary to develop a clear guidance for qualifying the software tools, to avoid any misinterpretation and difficulties when applying software related guidance to software tools. But also, it seemed necessary to export the tool qualification considerations outside of the "airborne domain". Therefore a tool vendor might apply a single qualification processes, independently of the domain. The goal is to benefit from a wider tool offer and to increase the tool quality.

For these reasons, the concept of "supplement" cannot be applied to "tools". The tool qualification considerations are the purpose of a new DO/ED document. As explained in section DO-178C Products the Tool Qualification Considerations document is used in conjunction with the domain related applicable document. To make the Tool Qualification Considerations document applicable, the domain-related applicable document should:

- Identify that the Tool Qualification Considerations document is applicable
- Define its own tool qualification criteria
- Define the tool qualification level (TQL-1 to TQL-5)

For airborne software, the new tool qualification criteria are explained in New Tool qualification criteria

Then, once the domain has defined the applicable criteria, the Tool Qualification Considerations document applies. Therefore, objectives to be satisfied for each TQL are defined, independently of the domain and of the qualification criteria.

As a first approach, the Tool Qualification Considerations document looks like DO-178C/ED-12 itself. This is because DO-178C/ED-12C was used as the basis of the development of this new document. But the text was adapted to be directly applicable to tools, and also to address all he tool aspects.

6. Make "thin" DO-248B/ED-94B?

Initial intent was that the new DO-178C/ED-12C would be clear, and thus would not require so much additional information as in DO-248B/ED-94B. It's true that it was possible to remove some information in DO-248B/ED-94B, but a better separation between the "guidance "from "additional information" also raised the need to switch some text from one document to another. As example, the "notes" in DO-178B/ED-12B are not part of guidance and can be moved to DO-248C/ED-94C.

DO-248C/ED-94C is applicable also to the ground domain. Therefore, changes were made in the existing text to be applicable to the two domains, and of course, cross reference to DO-278A/ED-109A were added.

6.1 DO-248B/ED-94B clean-up

DO-248B/ED-94B provides clarifications of DO-178B/ED-12B. Similarly, DO-248C/ED-94C will provide clarifications of DO-178C/ED-12C, and also of DO-278A/ED-109A.

A variety of editorial changes were necessary to keep the new documents consistent. The changes impact the terminology, the references (new sections of DO-178C/ED-12C but also adding references to DO-278A/ED-109A) and of course need to take into account when core document texts have been improved.

When editing DO-248B/ED-94B "Clarification of DO-178B/ED-12B", it was of course out of scope to make any change in DO-178B/ED-12B. But in some cases, DO-178B/ED-12B would be more understandable and easier to read if the DO-248B/ED-94B text could be directly inserted in the core document.

As a result, when possible, text was copied from DO-248B/ED-94B and equivalent information included in DO-178C/ED-12C. As a consequence such text becomes in most cases part of the guidance, and is no longer considered as "additional information".

Here is the list of information that doesn't appear in DO-248C/ED-94C because it has been moved into the core document:

- ***FAQ #1: Section 2 of DO-178B/ED-12B provides an introduction to the system aspects relating to software development and notes that guidelines were under development at the time of writing. Where are these system life cycle guidelines documented?***

  SAE ARP4754/ED-79 is a document that was not yet available at the time when DO-178B/ED-12B was issued. Section 2 now directly references ARP4754/ED-79 in the opening text of 2.0. Thus, this FAQ #1 is no longer needed

- ***FAQ #2: Throughout ED-12B reference is made to the system safety assessment process. Where can guidelines for this process be found?***

Section 2 now directly references ARP4754/ED-79 in the opening text of 2.0. ARP4754 then points to SAE ARP4761/ED-135 for all safety assessment process activities. Thus, this FAQ #2 is no longer needed.

- ***FAQ #3: What is meant by safety monitoring software experiencing transients in DO-178B/ED-12B section 2.3.2, paragraph 3?***

  Text referenced by this FAQ was updated to be more clear and is now in §2.4.2

- ***FAQ #6: What are the design description and verification activity objectives for a Level D system and why are there apparent inconsistencies in the objectives to be satisfied in Annex A?***

  The inconsistency has been removed, since some objectives for the development process (A2-4, 5 and 6) are no longer applicable to level D.

- ***FAQ #10: Are baselines allowed to be changed? Section 7.2.2.c states baselines should be protected from change, whereas section 7.2.4.c talks about changes to baselines.***

  Clarifications are included in the core text. (§7.2.4.d)

- ***FAQ #11: Is the "approved source" in section 7.2.7.a of DO-178B/ED-12B the previous approved product or is it the organization building the product?***

  Clarifications are included in the core text (§7.2.7.a), and the term “approved source” is defined in the glossary

- ***FAQ #12: What are the definitions of Control Categories 1 and 2 (CC1 and CC2)?***

  Clarifications are included in the core text (§7.3), and the term “control category” defined in the glossary

- ***FAQ #15: Is software certified as a stand-alone product?***

  Clarifications are included in the core text. (§10.0)

- ***FAQ #19: How does one determine if in-service problems indicate an inadequate process, and can one continue to pursue a service history means of compliance with some process inadequacies?***

  Section 12.3.4 on service history was enhanced and includes these clarifications. Information provided by this FAQ was also incorporated in DP#4

- ***FAQ #26: Does the fulfillment of "independence of multiple-version dissimilar software" (DO-178B/ED-12B section 12.3.3.1) supersede the independence requirements as defined in Annex A of DO-178B/ED-12B?***

  The main added value of the FAQ is included in form of a "note" in the core text (§12.3.2.1)

- ***FAQ #27: What is meant by "user-modifiable software"?***

  User-modifiable software has been clarified in DO-178C/ED-12C, and the term is now defined in the glossary

- ***FAQ #28: What is the value of removing dead code or unused variables?***

  Dead code, deactivated code and extraneous code were reworked in the core document, making this FAQ obsolete.

- ***FAQ #30: What does DO-178B/ED-12B section 2.6a(2) mean regarding system safety requirements addressing system anomalous behavior?***

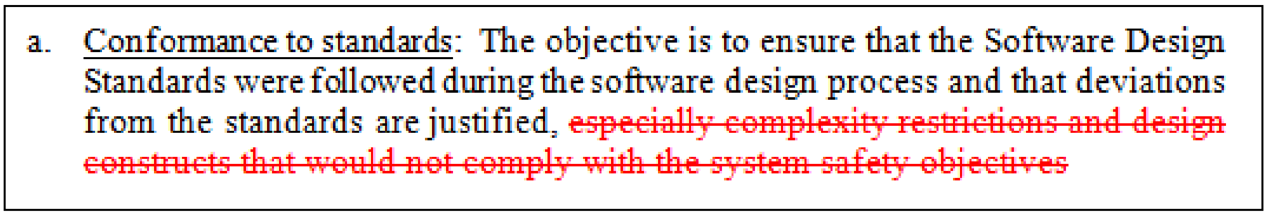
  The previous text of section 2.6 has been deleted. New text has been inserted into section 2 to address the prior content of section 2.6.

- ***FAQ #31: How does verification of product relate to "compiler acceptability"?***

  The main point of this FAQ is added in the form of a note in §4.4.2

- ***FAQ #33: Is it permissible to NOT meet the safety objectives by justifying any deviations from the design standards?***

As the clear answer is NO, the text is considered as trivial. However some confusing text in the core document was removed, e.g. section 6.3.3.e



a. 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, ~~especially complexity restrictions and design constructs that would not comply with the system safety objectives~~

- ***FAQ #34: What is the concept of independence as used in DO-178B/ED-12B?***

  Independence is discussed at length in a new discussion paper (DP#19)

- ***FAQ #38: What is the difference between Integration Process and Integration Testing?***

  The terms "Integration Process" and "Integration testing" are now defined in the glossary.

- ***FAQ #41: Why is Source Code to object code traceability required for Level A software?***

  Section 6.4.4.2 of the core document was updated, and a note added to clarify this topic.

  A part of this FAQ became wrong as it stated that "*Alternatively, structural coverage analysis can be performed on the object code, in which case traceability between source code to object code may not be needed*".

  Now the core document says that "*Independent of the code form on which the structural coverage analysis is performed, if the software level is A and a compiler, linker, or other means generates additional code that is not directly traceable to Source Code statements, then additional verification should be performed to establish the correctness of such generated code sequences"*

  DP#12 addresses also the same topic.

- ***FAQ #45: What is the relevance of the exception case stated in the definition of dead code?***

  "Extraneous code including dead code" replaces the former "dead code". More examples are provided in the glossary, and section 6.4.4.3 is changed in DO-178C.ED-12C

- ***FAQ #49: Where can current certification authority guidance regarding issues not covered in DO-178B/ED-12Bor expanding upon issues in DO-178B/ED-12B be found?***

The committee considered that the answer to this question may change over time, and is beyond the scope of the document. However FAQ#48 was modified and addresses the intent of this FAQ.

- ***FAQ #51: What is meant by the term "type design," as used in section 9.4 of DO-178B/ED-12B?***

  The definition of "type design" provided in this FAQ is now in the glossary

- ***FAQ #53: Do the data items need to be prepared and packaged as specified in section 11 of DO-178B/ED-12B?***

  Equivalent information is provided in a note in section 11.0 to clarify the packaging aspects.

- ***FAQ #61: What constitutes a development tool and when should it be qualified?***

  The term "development tool" is not used anymore. Some qualification criteria are provided in section 12.2. Rationale for this change is provided in a FAQ in the Tool Qualification Considerations document

- ***FAQ #66: What is the difference between certification, approval, and qualification?***

  Equivalent information is included in section 10.0. And the new FAQ#79 provides additional information on Technical Standard Order (TSO) process, and in Europe, the European Technical Standard Order (ETSO) process

- ***FAQ #71: What is the purpose of traceability, how much is required, and how is it documented? For example, is a matrix required or are other methods acceptable?***

  This FAQ was deleted since DO-178C/ED-12C now includes the "Trace Data" concept in section 5.5, 6.5, Table A-2, and Table A-6.

- ***DP #1: Verification Tool Selection Considerations***

  The term "verification tool" is not used anymore. Some qualification criteria are provided in section 12.2. Rationale for this change is provided in a FAQ in the Tool Qualification Document

- ***DP #2: The Relationship of DO-178B/ED-12B to the Code of Federal Regulations (CFRs) and Joint Aviation Requirements (JARS)***

  This DP is deleted since the FAA and EASA documents are changing and we don't know which documents will reference DO-178C and its supplements.

- ***DP #3: The Differences Between DO-178A/ED-12A and DO-178B/ED-12B Guidance for Meeting the Objective of Structural Coverage***

  Structural Coverage analysis is addressed by FAQ#44 that includes information from this DP.

- ***DP #7: Definition of Commonly Used Verification Terms***

  This DP is deleted since the terms are either no longer in DO-178C/ED-12C or have been included in the DO-178C/ED-12C glossary.

- ***DP #11: Qualification of a Tool Using Service History***

  Tool qualification aspects are addressed in DO-330/ED-215. This document includes a specific section (§11.4) providing guidance for using service history for qualifying a tool.

6.2 DO-248C/ED-94C improvements

This section identifies the changes beyond editorial that have been performed on existing FAQ/DP.

- ***FAQ #8: Can option-selectable software contain deactivated code?***

  Except for the answer "yes" to the question, all the text was replaced by references to the core text:

  - 2.5.4 description of option-selectable software
  - 4.2.h and 5.2.4 for designing for deactivated code

- ***FAQ #9: Do all high-level requirements require hardware/software integration testing? And, what does "To verify the interrelationships between software requirements and components" mean?***

  A change was made to clarify the answer to the first question by referencing section 6.4.1, which states that "*more than one test environment may be needed to satisfy the objectives for software testing.*"

- ***FAQ #24: What is the relationship between ARP4754A/ED-79A and DO-178C/ED-12C?***

  Slight adjustments were performed to correct the figure and make it consistent with ARP4754.

- ***FAQ #32: What are defensive programming practices?***

  This FAQ makes reference to the OOT supplement for the memory allocation issue. Additional text explains that programming practices don't supersede the need for requirements specifying the correct software response to abnormal conditions and inputs. (see Robustness)

- ***FAQ #35: What are low-level requirements and how may they be tested?***

  An example in this FAQ raised some comments (as usual for all examples) so it was decided to remove it. Other changes were for consistency with the addition of "Trace Data" in DO-178C/ED-12C.

- ***FAQ #36: What is the exact definition or interpretation of derived requirements in DO-178C/ED-12C and DO-278A/ED-109A?***

  As the definition of derived requirements was changed in the core document, (see Derived requirements and traceability), this FAQ provides example of the two "classes" of derived requirements

  - Requirements that do not trace directly to higher level of requirements: Those are typically based upon design, performance, or architectural decisions
  - Requirements that specify behavior that is in addition to the behavior specified by the system requirements or higher level requirements such as scaling limits on the fixed point arithmetic.

- ***FAQ #42: What needs to be considered when performing structural coverage at the object code level***

  As explained in this document (see Verification of additional code**)**, traceability between source code and object code was a "hidden objective" in DO-178B/ED-12B. To better address this subject, the core text was updated, FAQ#41 was removed, and additional information is now provided in this FAQ#42.

  It is now clearly established that the structural coverage analysis may be performed on source code or object code. As a result, this FAQ was rewritten. It focuses only on considerations when the structural coverage analysis is performed on object code.

- ***FAQ #54: Is the documentation required in DO-178C/ED-12C and DO-278A/ED-109A section 11 excessive, especially for small projects?***

  The answer is largely simplified as packaging has been clarified. So this FAQ only emphasizes that safety is more the issue than the project size.

- ***FAQ #65: What is meant by "equivalent software verification process activities" in DO-178C/ED-12C and DO-278A/ED-109A sections 12.3.2.4 (Tool Qualification for Multiple-Version Dissimilar Software) and 12.3.2.5 (Multiple Simulators and Verification)?***

  The existing FAQ was confusing and needed to be reworked.

  The question deals with the possible alleviation of the tool qualification process when qualifying dissimilar tools in the scope of Multiple-Version Dissimilar Software.

  The "equivalent software verification process activities" are linked to the confidence in the tools. It should be demonstrated that the use of dissimilar tools (with the proposed modified qualification process) provides the same confidence that the use of a single tool whose process is not modified.

  This FAQ just explains that it is possible to propose such reduction, but does not propose anything. It will be as usual in this approach, a case by case study.

- ***FAQ #67: What is analysis of data coupling and control coupling?***

  The main added value of the changes performed on this FAQ is to add an example of data and control coupling and how to achieve the coverage of this coupling with requirement-based tests. (See Structural coverage analysis)

- ***FAQ #73: Are timing measurements during testing sufficient or is a rigorous demonstration of worst-case timing necessary?***

  This FAQ was extended with additional details, providing a list of difficulties when determining the WCET, and references to the core text to provide a complete scope of this topic. Unfortunately, there is no "real" additional information.

- ***DP #4: Service History Rationale for DO-178C/ED-12C***

  This DP was completely rewritten to reflect the changes performed in section §12.3 of the core text. (see Service history).

  This paper discusses and provides example of:

  - The relevance of service history: Data about knowledge of the software functions, possible deactivated code, operating environment, configuration, the known errors and changes during the service history, and the process for collecting operating time or events.
  - The amount of service history: This section explains that time duration or events may be used, depending on the software. It also makes a connection, but without any values, between software level and certification credit claimed, with the amount of service history
  - Analysis of problems found during service history: This is not limited to analysis of functional problems. It should include the analysis of the efficiency of the problem reporting process through description of means and methods used. Then all problems, including process-related problems, and safety-related problems.

- The typical information to be provided in the PSAC to claim confidence of this alternate approach.

The scope of this DP is limited to DO-178C/ED-12C, as equivalent information has been included in the core text of DO-278A/ED-109A.

- ***DP #9: Assessment and Classification of Open Software Problems***

This DP is a full rewrite of the existing DP. The purpose of this DP is to provide additional information for a standardized assessment and classification methodology for open problem reports (OPRs) that are unresolved at the time of approval. The intent is to facilitate the approval of software with OPRs.

This DP includes the following information

- Define OPR classifications
- Perform OPR assessment
- Document OPR assessment results in the SAS
- Provide OPR assessment results to system integrator or system/equipment manufacturer to confirm potential effect at the system level
- Evaluate OPRs associated with reused software

Normally, this DP will supersede some well-know CRIs.

- ***DP #13: Discussion of Statement Coverage, Decision Coverage, and Modified Condition/Decision Coverage (MC/DC)***

This new definition of MC/DC as discussed in section 4.3.11 raised the need to update this DP. It provides other examples of statement, decision and MC/DC coverage, but it mainly adds new terms; the intent is to address some possible features of programming languages.

6.3 Text switched from DO-178B/ED-12B to DO-248C/ED-94C

Since a "note" in the core document is not part of the guidance, so it should appear instead in DO-248/ED-94. The initial intent was to replace the notes in the core document by some FAQ in DO-248/ED-94. However, including notes in the core text helps the reader to understand the material immediately, without needing to refer to a separate document.

But the approach of moving informative (non-normative) content from DO-178/ED-12 to DO-248/ED-94 was really applied only in one instance:

- ***FAQ #78: For software requirements expressed by logic equations, how many normal range test cases are necessary to verify the variable usage and the Boolean operators?***

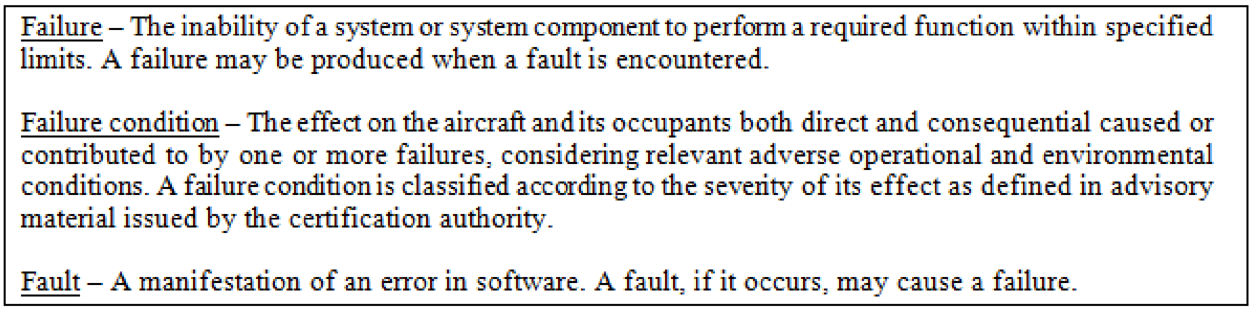
This FAQ replaces the note in section 6.4.1. But in other cases the DO-178C/ED-12C text includes informative content to avoid possible misinterpretations (see Note 6.4.2.1.d on test cases).

6.4 New FAQ and DP

Several FAQ and DP were added mainly to provide additional information about existing or new content in DO-178C. But several others discuss issues not addressed in the core document.

- ***FAQ #77: The Software Requirements Data are described by DO-178C/ED-12C and DO-278A/ED-109A section 11.9. What is meant in step 11.9 g "Failure detection and safety monitoring requirements"?***

As part of the revision work on section 2, several terms were added to the glossary, with the same definitions as in the system documents (ARP4754/ED-79 and ARP4761/ED-135):



Failure – The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered.

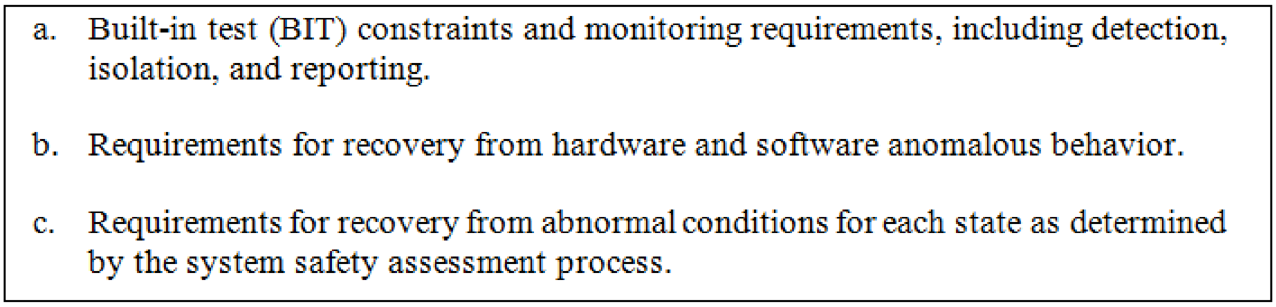
Failure condition – The effect on the aircraft and its occupants both direct and consequential caused or contributed to by one or more failures, considering relevant adverse operational and environmental conditions. A failure condition is classified according to the severity of its effect as defined in advisory material issued by the certification authority.

Fault – A manifestation of an error in software. A fault, if it occurs, may cause a failure.

It is important for these definitions to be consistent and to reflect that the requirements on failure detection and "safety monitoring" are connected to the system safety assessment.

However, it was decided to leave the text of section 11.9.g unchanged, but to add this new FAQ. This FAQ first defines the scope of these requirements: "Software may be used to detect, monitor, and control failures;"

And in this context the FAQ provides some typical examples:



a. Built-in test (BIT) constraints and monitoring requirements, including detection, isolation, and reporting.

b. Requirements for recovery from hardware and software anomalous behavior.

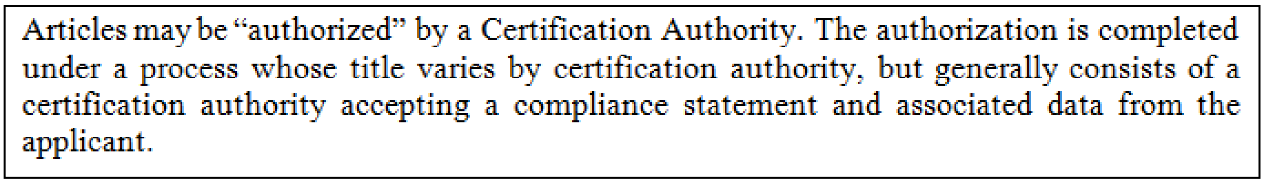
c. Requirements for recovery from abnormal conditions for each state as determined by the system safety assessment process.

- ***FAQ #79: Can an applicant for aircraft, engine or propeller take credit for DO-178C/ED-12C compliance found under an article approval (that is, Technical Standard Order or European Technical Standard Order)?***

  Many products that include software receive authorization under the TSO, ETSO, or equivalent process. The relationship of this software approval to the Type Certification process is not discussed in DO-178C/ED-12C.

  This has sometimes led to confusion on responsibility for software approval between the TSO Certification Authority, the aircraft manufacturer, and the aircraft manufacturer's certification authority.

  So this FAQ describes the typical process to "authorize" the article:



Articles may be "authorized" by a Certification Authority. The authorization is completed under a process whose title varies by certification authority, but generally consists of a certification authority accepting a compliance statement and associated data from the applicant.

- ***FAQ #80: What needs to be considered when using inlining?***

  This FAQ was developed by the Object Oriented Techniques subgroup. But since this topic is not specific to that domain, it was included in DO-248C/ED-94C.

  Thus FAQ identifies how several issues are affected by the use of the inlining compiler directive, including WCET, memory and stack usage, source code to object code traceability and data and control coupling. Thus an analysis of the object code to identify the impact of inlining is recommended.

- ***FAQ#81 What aspects should be considered when there is only one level of requirements (or if high-level requirements and low-level requirements are merged)?***

  Development processes generally produce at least two levels of requirements: HLRs that represent "what" and LLRs that represent "how". It is allowed, as explained in section 5, to produce a single level of requirements from the system requirements. But this approach raises some concerns from the certification authorities. So a new FAQ, initiated by European certification authorities, doesn't recommend combining or merging HLR and LLR. The main concern is possible gaps in the requirement refinement, which prevent a sufficient traceability analysis and thus compliance with verification objectives.

- ***FAQ#82 "If pseudocode be used as part of the low-level requirements, what issues need to be addressed?***

  The term "pseudo-code" is not used in the core document. But the term "algorithm" is defined in the glossary, and is part of the design data as defined in section 11.10: The design data "should include: a detailed description of how the software satisfies the specified high-level requirements, including **algorithms**, data structures, and how software requirements are allocated to processors and tasks.”

“Algorithms” may be expressed in “pseudo-code”; that does not contradict the definition provided in this FAQ!

> Pseudocode is a textual description of a computer programming algorithm that uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are not essential for human understanding of the algorithm (for example, variable declarations, system-specific code, and subroutines).

However, this FAQ, initiated by the European certification authorities, discusses some of the difficulties in satisfying some objectives when using pseudo-code to describe the LLR. The mains concerns are the difficulties in identifying unintended functions in LLR, and the reduction of the capability of low-level tests to detect incorrect or missing functionality.

Therefore, the intent of this FAQ is to discourage the systematic use of pseudo code for all or most low-level requirements.

- ***FAQ#83 “Should compiler errata be considered?”***

The purpose of this FAQ is to clarify that “known tool problems and limitations” discussed in section 4.4.1.f are also applicable to compilers. (See Software Development Environment).

It should be noted also that analysis of compiler warnings was also added as an activity of the verification of the outputs of the integration process (Errors and Inconsistencies number 5)

- ***FAQ#84: How can all Level D (AL 5) objectives be met if low-level requirements and Source Code are not required?***

This FAQ is concerned with the effects of changing the development process objectives to level D (see Objectives for Level D software). At the time this issue was discussed, the core text was already approved. It was then decided to add this new FAQ to address potential inconsistencies of the core document.

This FAQ explains that the LLR and source code may still need to be produced, because at least two other objectives need these artifacts:

- table A-4 objective 13 on partitioning
- table A-8 objective 4 on archive and retrieval: This objective requires the regeneration of the software.

- ***DP #16: Cache Management and DP #17: Usage of Floating-Point Arithmetic***

These DP includes some CRI often identified by projects with respect to the "consistency and accuracy of source code" objective (See New topics in accuracy and consistency of source code).

- ***DP #18: Service Experience Rationale for DO-278A/ED-109A***

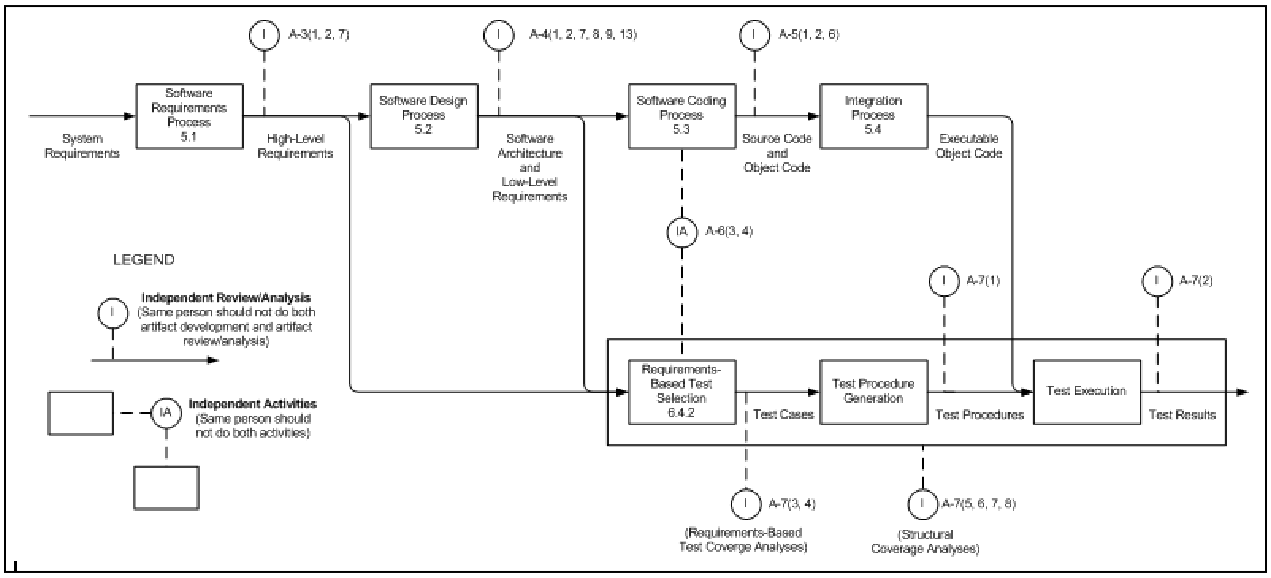
This paper clarifies guidance from DO-278A/ED-109A on considerations in using service experience as an alternate means of compliance. Discussion or examples of each topic are provided.

DP#4 addresses the same topic for DO-178C/ED-12C. Though minor, some differences exist between the two domains, and it was decided to keep 2 separate DPs.

- ***DP #19: Independence in DO-178C/ED-12C and DO-278A/ED-109A***

This DP replaces the previous FAQ#34; it explains the purpose of independence: "The purpose of independence is to avoid having a misinterpretation of requirements by a single means carry through both the design and the verification of a function".

Through the following figure it also provides a complete view of applying independence, and specifies each objective where independence may be required.



I
A-3(1, 2, 7)
I
A-4(1, 2, 7, 8, 9, 13)
I
A-5(1, 2, 6)
Software Requirements Process 5.1
System Requirements
High-Level Requirements
Software Design Process 5.2
Software Architecture and Low-Level Requirements
Software Coding Process 5.3
Source Code and Object Code
Integration Process 5.4
Executable Object Code
IA
A-6(3, 4)
LEGEND
I
Independent Review/Analysis
(Same person should not do both artifact development and artifact review/analysis)
IA
Independent Activities
(Same person should not do both activities)
I
A-7(1)
I
A-7(2)
Requirements-Based Test Selection 6.4.2
Test Cases
Test Procedure Generation
Test Procedures
Test Execution
Test Results
I
A-7(3, 4)
(Requirements-Based Test Coverge Analyses)
I
A-7(5, 6, 7, 8)
(Structural Coverage Analyses)

After discussion, it was finally agreed that, to satisfy the independence criteria for the objectives on compliance of executable object code with LLR, two possible approaches are acceptable:

- The same person(s) could develop the low-level requirements and the Source Code, provided another person(s) develops the test cases from those low-level requirements, or
- The same person(s) could develop the low-level requirements and their associated test cases, provided that another person(s) develops the Source Code.
-

- ***DP #20: Parameter Data Items and Adaptation Data Items***

  As PDI is a new topic, it was considered that some clarifications need to be provided for a better understanding and a consistent application. So this DP includes the following topics:

    - What is a parameter data item?
    - Which considerations apply with respect to compatibility between Executable Object Code and Parameter Data Item Files?
    - What does the fourth bullet of section DO-178C/ED-12C section 6.6 mean?
    - How is a separately verifiable Parameter Data Item File verified?
    - How is PDI related to option-selectable software, user-modifiable software, and field-loadable software?
    - What is the rationale for the "normal range testing" and the "robustness of the EOC" defined in section 6.6?
    - What is the rationale for assigning the PDI the same software level as the component using it?

- ***DP#21: Clarification on Single Event Upset (SEU) as It Relates to Software***

  SEU is a new topic identified in the core text, and the means to mitigate the SEUs becomes an additional consideration to be addressed in the PSAC. This DP includes the following considerations. (See Single Event Upsets).

    - What is a Single Event Upset (SEU)?
    - When is it necessary to apply SEU mitigation measures in software?
    - What are some of the SEU mitigation techniques that can be implemented in software?
    - What other options are available for SEU mitigation?
    - What other information is available on single event upset (SEU) mitigation for avionics?

… for any questions, to ask for a DO-178C/ED-12C training, or to propose additional inputs and improvements to this document, please contact :

Frédéric POTHON – ACG SOLUTIONS

(+33) 04.67.60.94.87 – (+33) 06.21.69.26.80

frederic.pothon@acg-solutions.fr

www.acg-solutions.fr