EASA	**NOTIFICATION OF A PROPOSAL TO ISSUE A CERTIFICATION MEMORANDUM**
	EASA Proposed CM No.: **EASA Proposed CM - SWCEH – 002 Issue: 01** **Issue Date: 10th of February 2011** **Issued by: Software & Complex Electronic Hardware section** **Approved by: Head of Certification Experts Department** **Effective Date: [Standard date = 7 days after final CM Issue date]** **Regulatory Requirement(s): CS 25.1301 and 1309 for Large Aeroplanes, CS23.1301 and 23.1309 for Small Aeroplanes, CS27.1301 and 27.1309 for Small Rotorcraft, CS29.1301 and 29.1309 for Large Rotorcraft, CS E-50 (d,f) for engines, CS-P, CS-APU and CS-ETSO.**

In accordance with the EASA Certification Memoranda procedural guidelines, the Agency proposes to issue an EASA Certification Memorandum (CM) on the subject identified below.

All interested persons may send their comments, referencing the EASA Proposed CM Number above, to the e-mail address specified in the "Remarks" section, prior to the indicated closing date for consultation.

EASA Certification Memoranda clarify the Agency's general course of action on specific certification items. They are intended to provide guidance on a particular subject and, as non-binding material, may provide complementary information and guidance for compliance demonstration with current standards. Certification Memoranda are provided for information purposes only and must not be misconstrued as formally adopted Acceptable Means of Compliance (AMC) or as Guidance Material (GM). Certification Memoranda are not intended to introduce new certification requirements or to modify existing certification requirements and do not constitute any legal obligation.

EASA Certification Memoranda are living documents, into which either additional criteria or additional issues can be incorporated as soon as a need is identified by EASA.

Subject

Software Aspects of Certification

Log of Issues

Issue	Issue date	Change description
01	09.02.2011	Initial issue.

Table of Contents

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

The purpose of this Certification Memorandum is to provide specific guidance material to the applicant on various aspects complementary to ED-12B/DO-178B.

1.2 REGULATORY REFERENCES & REQUIREMENTS

It is intended that the following reference materials be used in conjunction with this Certification Memorandum:

Reference	Title	Code	Issue	Date
ED-12B / DO-178B	Software Considerations In Airborne Systems and Equipment Certification	EUROCAE ED-12B RTCA DO-178B	B	December 1992
ED-94B / DO-248B	Final report for clarification of ED12B / DO178B "Software Considerations in Airborne Systems and Equipment Certification".	EUROCAE ED-94B RTCA DO-248B	B	October 2001
ED-79 / ARP4754	Certification Considerations for Highly Integrated or Complex Aircraft Systems.	EUROCAE ED-79 SAE ARP4754	-	November 1996
AMC 20-115B	Recognition of EUROCAE ED-12B / RTCA DO-178B	AMC-20	Initial	November 2003

1.3 ABBREVIATIONS

The following abbreviations are used in this Certification Memorandum:

Abbreviation	Meaning
A/C	Aircraft
ABC	Assembly Branch Coverage
AMC	Acceptable Means of Compliance
CAST	Certification Authorities Software Team
CEH	Complex Electronic Hardware
CF	Configuration File
CID	Configuration Index Document
CM	Certification Memorandum
COTS	Commercial Off-the-shelf
CRC	Cyclic Redundancy Check
CRI	Certification Review Item
CS	Certification Specification(s)
CSCI	Computer Software Configuration Item
DAL	Development Assurance Level
DOA	Design Organisation Approval
EASA	European Aviation Safety Agency
EIS	Entry Into Service
FAA	Federal Aviation Administration
FAQ	Frequently Asked Question
FHA	Functional Hazard Analysis
FLS	Field-Loadable Software
GM	Guidance Material
HLR	High-level Requirement
IFCA	Instructions for Continued Airworthiness
IMA	Integrated Modular Avionics
JAA	Joint Aviation Authorities (predecessor of EASA)
LLR	Low-level Requirement
LOI	Level of Involvement
MCDC	Modified Condition Decision Coverage
MEL	Minimum Equipment List
OCC	Object Code Coverage
OOT	Object-oriented Technique

Abbreviation	Meaning
OPR	Open Problem Report
P/N	Part Number
PCM	Project Certification Manager
PDS	Previously-Developed Software
PID	Project Information Document
PSAC	Plan for Software Aspects of Certification
PSSA	Preliminary System Safety Analysis
RBT	Requirement-based Testing
RTC	Restricted Type Certificate
SAS	Software Accomplishment Summary
SCI	Software Configuration Index
SCMP	Software Configuration Management Plan
SDP	Software Development Plan
SECI	Software Life Cycle Environment Configuration Index
SOI	Stage of Involvement
SQAP	Software Quality Assurance Plan
SW	Software
SW/CEH	Software / Complex Electronic Hwr
STC	Supplemental Type Certificate
SVP	Software Verification Plan
TAS	Tool Accomplishment Summary
TC	Type Certificate
TGL	Temporary Guidance Leaflet
TOR	Tool Operational Requirements
TQP	Tool Qualification Plan

1.4 DEFINITIONS

Some terms of this CM are defined below; however, in order to improve the readability of this CM, some sections contain specific definitions (e.g. section 2). The reader may also need to refer to the definitions contained in certain Eurocae standards (e.g. ED-12B/DO-178B) as they are not repeated below.

Definition	Meaning
Aeronautical Data (ED-76/DO-200A)	Data used for aeronautical applications such as navigation, flight planning, flight simulators, terrain awareness and other purposes, which comprises navigation data and terrain and obstacle data.
Aeronautical Database (ED-76/DO-200A)	An Aeronautical Database is any data that is stored electronically in a system that supports airborne or ground based aeronautical applications. An Aeronautical Database may be updated at regular intervals.
Configuration Files	Files embedding parameters used by an operational software program as computational data, or to activate / deactivate software components (e.g. to adapt the software to one of several aircraft/engine configurations). The terms 'registry' or 'definition file' are sometimes used for a Configuration File. Configuration files such as symbology data, bus specifications or aircraft/engine configuration files are segregated from the rest of the embedded software for modularity and portability purposes.
Database (ED-12B/DO-178B)	A set of data, part or the whole of another set of data, consisting of at least one file that is sufficient for a given purpose or for a given data processing system.
Field-loadable software	software that can be loaded without removal of the equipment from the installation. Field-loadable software can refer to either executable code or data. (Refer to ED-12B / DO-178B, Section 2.5.)
Higher-level Requirements	In order to produce either a set of Formalized Requirements or a Formalized Design, a set of requirements at a higher-level of abstraction is needed in order to capture the requirements for the Formalized Requirements or Formalized Design and to describe what the resulting formalized item should contain. Such requirements are therefore known hereafter in this Certification Memorandum as 'higher-level requirements'. The data item(s) that act as higher-level requirements should be identified during the planning stage.
Option-selectable software	Software that contains approved and validated components and combinations of components that may be activated by the user, either through selection by the flight crew or activation by ground personnel. (Refer to ED-12B / DO-178B, Section 2.4.)
SW/CEH Group	SW/CEH panel plus SW/CEH assistant specialists where applicable. NOTE: the size of the SW/CEH group can vary, depending on the size of the certification project and the number of pieces of digital equipment to be qualified. For small projects, the SW/CEH group (and panel) may be limited to one person taking charge of the complete spectrum of tasks and responsibilities described in this section (including coordination).
SW/CEH Panel	EASA nominated specialist(s).

Definition	Meaning
User-Modifiable Software	As the term is used in ED-12B / DO-178B, is software intended for modification by the aircraft operator without review by the certification authority, the airframe manufacturer, or the equipment vendor. Modifications by the user may include modifications to data, modifications to executable code, or both. (Refer to ED-12B / DO-178B, Section 2.4.) NOTE: Modifications by the user to user-modifiable software may include modifications to data, modifications to executable code, or both, if within the modification constraints established during the original certification program.
Validation	The determination that the requirements for a product are sufficiently correct and complete.
Verification	The evaluation of an implementation of requirements to determine that they have been met.

2 BACKGROUND

Current aircraft systems include pieces of digital equipment that contain software components. Compliance with CS 25.1301 and 1309[1] is partly addressed through development assurance activities conducted on the system itself. Additionally, in accordance with AMC 20-115B, the applicant may choose EUROCAE ED-12B/RTCA DO-178B as an approved method to secure software approval.

The EUROCAE ED-12B /RTCA DO-178B document does not, however, provide sufficient guidance regarding some important aspects such as the software review process, field loadable software, user-modifiable software, software changes in legacy systems, software tool qualification, software change classification or the re-use of life cycle data. Other items that require further guidance include the use of configuration files, object-oriented technology, the use of assembly branch coverage in structural coverage analysis, the management of open problem reports, the oversight of suppliers and Formalised and Model Based requirements and designs.

The aim of this Certification Memorandum is to provide additional guidelines to the applicant on these aspects.

2.1 COMPARISON BETWEEN THE CONTENTS OF THIS DOCUMENT AND THE CONTENT OF EXISTING FAA ORDERS

The format of this Certification Memorandum in terms of the order of the sections is intended to harmonise this EASA guidance material with the existing FAA guidance material. Sections 3 – 14 of this Certification Memorandum correspond to chapters 1 – 12 of FAA Order 8110.49. Sections 15 – 18 of this Certification Memorandum correspond to chapters 1 – 4 of FAA Notice 8110.110. This may facilitate recognition of the various sections of the guidance when certification credit is requested.

Applicants should note, however, that apart from some minor differences in wording and paragraph numbering, in some cases, the content of the guidance contained in this Certification Memorandum is different from the guidance contained in FAA Order 8110.49 and Notice N8110.110. The major differences are described below.

a) The following sections of this Certification Memorandum contain some significant differences from the guidance provided by the equivalent chapters of the FAA Orders and Notices that exist at the time of publication of this document –
   - Section 5 – 5.3.1, Organization, Role and Level of Involvement of EASA and Applicants in Software Projects – these sub-sections differ from the contents of chapter 3 of FAA Order 8110.49 in that they describe the role of the EASA software panel and include the determination and documentation by an applicant of their level of involvement in the software of each system on an aircraft.
   - Section 6 - this section has no content in this Certification Memorandum, whereas chapter 4 of FAA Order 8110.49 covers Software Conformity Inspection.
   - Section 7 - the note on MEL in chapter 5 of FAA Order 8110.49 has been deleted from this section, as the MEL considerations are developed according to specific EASA guidance.

[1] This applies for Large Aeroplanes. For other products, please refer to CS23.1301 and 23.1309 for Small Aeroplanes, CS27.1301 and 27.1309 for Small Rotorcraft, CS29.1301 and 29.1309 for Large Rotorcraft, CS E-50 (d,f) for engines, CS-P, CS-APU and CS-ETSO.

- Section 8 - this section has no content in this Certification Memorandum, whereas chapter 6 of 8110.49 covers Approval of Field-Loadable Software (FLS) by Finding Identicality Through the Parts Manufacturer Approval (PMA) Process.
- Section 13 – this section on the Oversight of Software Change Impact Analyses Used to Classify Software Changes as Major or Minor differs considerably from the content of chapter 11 of 8110.49.
- Section 16.1 -16.7 - Management of Open Problem Reports – the contents of these parts of this section differ from the contents of chapter 2 of Notice 8110.110, which is entitled Software Problem Reporting.
- Section 17 - this section on Embedded Software Configuration Files differs from chapter 3 of Notice 8110.110, which is entitled Assuring System Databases and Aeronautical Databases.
- Section 18.2 2) - the wording of this sub-section differs from the wording in chapter 4 of Notice 8110.110 on Managing the Software Development and Verification Environment.

b) The following sections of the guidance of this Certification Memorandum do not correspond to the contents of the existing FAA Orders or Notices, but they do correspond to the contents of existing CAST Papers -

- Section 20, The Use of Object Code Coverage for Equivalence To Modified Condition Decision Coverage (CAST 17).
- Section 21, Merging High-Level and Low-Level Requirements (CAST 15).
- Section 22, Clarification of Structural Coverage Analyses of Data Coupling and Control Coupling (CAST 19).

c) The sections of this Certification Memorandum whose contents neither directly correspond to the contents of the existing FAA Orders or Notices available at the time of publication of this document nor to the contents of any existing CAST Papers are as follows –

- Section 19, The Use of Object-Oriented Techniques at the Design or Source Code Level.
- Section 23, The Validation and Verification of Formalized and Model-based Software Requirements and Designs.
- Section 24, The Use of Pseudo-Code as Low-level requirements.
- Section 25, Stack Overflows.

3 EASA CERTIFICATION POLICY

3.1 EASA POLICY

AMC 20-115B recognises Eurocae ED-12B / RTCA DO-178B as an acceptable means of compliance with EASA Certification Specifications. Accordingly, EASA policy on software aspects of certification is to permit applicants to use ED-12B / DO-178B as an acceptable means of compliance against chapters 1301 and 1309 of the various Certification Specifications or CS-E 50 (d,f), supplemented by the guidance provided in this Certification Memorandum.

3.2 WHOM THIS CERTIFICATION MEMORANDUM AFFECTS

The guidance contained in this Certification Memorandum applies to any applicants seeking approval from EASA for software embedded in aircraft systems or engines that is intended to comply with ED-12B / DO-178B. It also applies to any personnel involved in the ED-12B / DO-178B activities related to the airborne software of those applicants.

For TCs and STCs, applicants should ensure that they use the appropriate version of the Certification Memorandum called up in the applicable CRI.

For an ETSO, the applicant may decide to take into account all or part of this guidance contained herein, and may substantiate the details of their compliance in specific documentation (i.e. Declaration of Design and Performance, Software Accomplishment Summary, Hardware Accomplishment Summary or equivalent). Caution should be taken as the content of Certification Memoranda may have changed by the time the equipment is installed in the Aircraft/Engine. In any case, the installed equipment should finally comply with the Aircraft/Engine Certification Basis (including certain Certification Review Items).

When this Certification Memorandum is used outside of the scope of a TC, STC or ETSO (e.g. for pre-consultancy, pre-application , etc.), this guidance is provided for information only and caution should be taken as the content of the Certification Memorandum may have changed by the time of the application.

3.3 BACKGROUND

This Certification Memorandum was originally extracted from JAA leaflet n°5 (JAA interim guidance material on Software Aspects of Certification in addition to Eurocae ED-12B / RTCA DO-178B) and updated to take EASA requirements and procedures into account. It also incorporates some material that was formerly provided in separate Certification Memoranda and Certification Authorities Software Team (CAST) papers.

It should be noted that the term ‘Type Certificate’ (TC) in this Certification Memorandum refers both to Type Certificates (TCs) and to Restricted Type Certificates (RTCs).

3.4 THE USE OF EUROCAE ED-94B / DO-248B CLARIFICATIONS

The purpose of ED-94B / DO-248B is to provide clarification of the guidance material in ED-12B / DO-178B.

ED-94B / DO-248B may be used for any or all of the following purposes:

- Resolution of content errors in ED-12B / DO-178B.
- Clarification of a specific section or topic of ED-12B / DO-178B.
- Resolution of an inconsistency between ED-12B / DO-178B and any other relevant civil aviation standards.

4 GUIDELINES FOR THE SOFTWARE REVIEW PROCESS

4.1 PURPOSE

This section provides guidelines for conducting software reviews during the software development life cycle of airborne systems and equipment that are developed to meet the objectives of ED-12B/DO-178B. The guidelines below are used by SW/CEH experts and may be used by the applicant as indicated in section 4.3.

4.2 DEFINITIONS

For the purpose of this section, the following definitions apply:

Review is the act of inspecting or examining software life cycle data, software project progress and records, and other evidence produced with the intent of finding compliance with ED-12B/DO-178B objectives. Review is an encompassing term and may consist of a combination of reading, interviewing project personnel, witnessing activities, sampling data, and participating in presentations. A review may be conducted at one's own desk, at an applicant's facility, or at an applicant's supplier's facility.

Sampling is the process of selecting a representative set of software life cycle data for inspection or analysis to attempt to determine the compliance of all the software life cycle data developed up to that point in time in the project. Sampling is the primary means of assessing the compliance of the software processes and data. Examples of sampling may include any or all of the following:

- An inspection of the traceability from system requirements to software high-level requirements to software low-level requirements to source code and from software requirements to test cases and procedures to test results.
- A review of any analyses used to determine the system safety classification and the software level or of any reviews or analyses used to meet any ED-12B/DO-178B objective (e.g., timing analysis or code review).
- An examination of the structural coverage of multiple samples of source code modules.
- An examination of multiple samples of software quality assurance records and configuration management records.

Finding is the identification of a failure to show compliance with one or more of the objectives of ED-12B/DO-178B or of this Certification Memorandum.

Action: is the description of the activity to be performed by the applicant/supplier in order to resolve a finding or any other deficiency detected by the auditor. By default, actions should be completed and closed before approval.

Observation is the identification of a potential software life cycle process improvement.

Recommendation: is the description of the activity to be performed by the applicant/supplier in order to resolve an observation identified by the auditor. Implementation of recommendations is not mandatory prior to approval.

4.3 SCOPE

a. Section 9 of ED-12B / DO-178B describes the certification liaison process. The certification liaison process is the vehicle to establish communication and understanding between the applicant and the certification authorities. Sections 9.2 and 10.3 of ED-12B / DO-178B state that the certification authority may review the software life cycle processes and data to assess compliance with ED-12B / DO-178B. This section does not change the intent of ED-12B / DO-178B with regard to the software review process, but it clarifies the application of ED-12B / DO-178B.

b. The applicant should perform an equivalent software review process meeting the same objectives as described in this section. The review reports are usually requested by EASA.

c. Although desktop reviews may be used to successfully accomplish the software review process, this section of the Certification Memorandum primarily focuses on on-site reviews. The desktop review uses similar techniques as the on-site review but does not have the advantages of being on-site (e.g., access to software personnel, access to all automation, access to the test set-up). Both on-site and desktop reviews may be delegated to the properly authorised staff responsible for conducting certification-related activities. Practical arrangements with the software developer for on-site reviews by certification authorities should include:

(1) Agreement on the type of review(s) that will be conducted (i.e., planning, development, verification, or final certification)

(2) Agreement on date(s) and location(s) of the review(s).

(3) Identification of the certification authority personnel involved.

(4) Identification of any staff responsible for conducting certification-related activities who are involved.

(5) Development of the agenda(s) and expectations.

(6) Listing of software data to be made available (both prior to the review(s) and at the review(s)).

(7) Clarification of the procedures intended to be used.

(8) Identification of any required resources.

(9) Specification of date(s) and means for communicating review results (may include corrective actions and other required post-review activities).

d. The objectives of the software review process are found in paragraph 4.4 of this section. Paragraph 4.5 of this section primarily addresses the integration of the software review process with the software development life cycle. Paragraph 4.5 also identifies the four types of reviews and the software life cycle data and data assessment criteria for each type. Paragraph 4.6 of this section addresses additional considerations for the software review process. Paragraph 4.7 of this section provides guidelines for preparing, conducting, and documenting a software review.

4.4 OBJECTIVES OF THE SOFTWARE REVIEW PROCESS

a. The certification authorities may review the software life cycle processes and associated data at their discretion to obtain assurance that a software product submitted as part of a certification application complies with the certification basis and the objectives of ED-12B / DO-178B. The software review process assists both the certification authorities and the applicant in determining whether a particular project will meet the certification basis and ED-12B / DO-178B objectives by providing:

(1) Timely technical interpretation of the certification basis, the ED-12B / DO-178B objectives and CRIs.

(2) Visibility into the compliance of the implementation and the applicable data.

(3) Objective evidence that the software project adheres to its approved software plans and procedures.

(4) The opportunity for the certification authorities to monitor the activities of staff

responsible for conducting certification-related activities under the applicant's DOA system.

b. The amount of certification authority involvement in a software project should be determined and documented as early as possible in the project life cycle. The type and number of software reviews will depend on the software level of the project, the amount and quality of support from the staff responsible for conducting certification-related activities, the experience and history of the applicant and/or software developer, any history of service difficulties, and several other factors. Section 5 of this Certification Memorandum provides specific guidelines for determining the EASA level of involvement.

4.5 INTERACTION BETWEEN THE SOFTWARE REVIEW PROCESS AND THE SOFTWARE LIFE CYCLE

a. The review process should begin early in the software life cycle. Early certification authority involvement will mitigate the risk that the system, software, and planning decisions will not comply with the ED-12B / DO-178B objectives. This requires timely communication between the applicant and the certification authorities regarding those planning decisions that may impact the software product and processes. Typically, the development of software associated with an aircraft/ engine component may take several months or years. Since the guidance of ED-12B / DO-178B is process orientated, then if it is to be meaningful, the review process should be integrated throughout the software life cycle. This means that regular contact between the applicant and certification authorities should be established. This contact should provide gradually increasing confidence in the software life cycle processes and in the resultant product to both the applicant and the certification authorities. The four types of reviews are described as follows:

(1) A software planning review should be conducted when the initial software planning process is complete (i.e., when most of the plans and standards are complete and reviewed). This review is commonly referred to as stage of involvement (SOI) #1.

(2) A software development review should be conducted when at least 75% of the software development data (i.e., requirements, design, and code) are complete and reviewed. This review is commonly referred to as SOI #2.

(3) A software verification review should be conducted when at least 75% of the software verification and testing data are complete and reviewed. This review is commonly referred to as SOI #3.

(4) A final certification software review should be conducted after the final software build is completed, the software verification is completed, a (preliminary) software conformity review has been conducted, and the software product is ready for formal system certification approval. This review is commonly referred to as SOI #4.

b. The availability of software life cycle data does not imply that the data is always complete. However, the data should be sufficiently mature so that a reasonable review can be conducted. Similarly, not all transition criteria may necessarily be complete at that time in the project, but sufficient transition criteria evidence should exist to ensure they are being applied to the project.

c. Discussions between the applicant and the certification authorities should occur early in the project life cycle and should determine the types, need, number, depth, and format of the software reviews. For the purpose of this section of the Certification Memorandum, four reviews are identified to assess compliance with ED-12B / DO-178B objectives.

d. The following paragraphs define the basic goals of each of the four types of software reviews, the criteria for each type of review (e.g., type and availability of data, type of transition criteria), and the appropriate evaluation criteria. Paragraph 4.6 of this Certification Memorandum identifies additional considerations that may impact the type and timing of reviews.

4.5.1 Software Planning Review

a. ***Identification of the Software Planning Review.*** The software planning process is the initial process in the software life cycle for any software project. The planning process establishes the various software plans, standards, procedures, activities, methods, and tools required to develop, verify, control, assure, and produce the software life cycle data. The intent of the software planning review is to determine whether the applicant's plans and standards provide an acceptable means for complying with the objectives of ED-12B / DO-178B. This review can also reduce the risk of an applicant producing a software product that is inconsistent with the certification criteria and which will not support the continued airworthiness requirements of the product. The software planning review should take place after the initial completion of the software planning process. Although the software planning process may continue throughout the software life cycle, and plans and standards may change as the project progresses, it is generally considered complete when the associated initial transition criteria are satisfied. The following transition criteria are indicative of typical software planning process completion criteria:

(1) Software plans and standards have been internally reviewed, based on company specified criteria and deficiencies have been resolved.

(2) Software plans and standards have been evaluated by software quality assurance and deficiencies have been resolved.

(3) Software plans and standards have been approved and placed under configuration control.

(4) The objectives of ED-12B / DO-178B, Annex A, Table A-1 have been satisfied.

b. ***Data Required for the Software Planning Review.*** The applicant should make the software plans and standards shown in Table 4-1 available to the certification authorities. The supporting software data should be under configuration control, appropriate for the software level, prior to the software planning review.

Software Data	ED-12B / DO-178B Section
Plan for Software Aspects of Certification	11.1
Software Development Plan	11.2
Software Verification Plan	11.3
Software Configuration Management Plan	11.4
Software Quality Assurance Plan	11.5
*Software Requirements, Design, and Code Standards	11.6, 11.7, 11.8
Tool Qualification Plans, if applicable	12.2, 12.2.3.1
*Software Quality Assurance Records as applied to the planning activities	4.6, 11.19

* Not required for Level D, per ED-12B / DO-178B, Annex A, Table A-1.

Table 4-1 Data Availability for Software Planning Review

c. ***Evaluation Criteria for the Software Planning Review.*** The objectives that apply to planning in ED-12B / DO-178B Annex A, Tables A-1 (all objectives), A-8 (objectives 1-4), A-9 (objective 1), and A-10 (objectives 1-2), should be used as the evaluation criteria for the software planning review. The plans should also be evaluated to ensure that, when they are followed, all applicable RTCA/DO-178B objectives would be satisfied. Additionally, the applicant's safety assessment, failure conditions, and software level(s) should be assessed. Additionally, the applicant's safety assessment, failure conditions, and software level(s) should be assessed. The relevance of the software plans and standards to the software level should also be evaluated.

4.5.2 Software Development Review

a. ***Identification of the Software Development Review.*** The software development processes are the software requirements, design, code, and integration processes. The development processes are supported by the integral processes of software verification, configuration management, quality assurance, and certification liaison processes. Therefore, the software development review should assess the effective implementation of the applicant's plans and standards through examination of the software life cycle data, particularly the software development data and integral processes' data associated with it. During this review, the applicant and certification authority representatives may come to agreement on changes to or deviations from plans and standards that are discovered during the review. Before conducting a software development review, the software development data should be sufficiently complete and mature to ensure that enough evidence exists that the developer is complying with their approved plans, standards, and transition criteria. The following are typical criteria for a sufficiently mature software development process:

(1) High-level requirements are documented, reviewed, and traceable to system requirements.

(2) The software architecture is defined, and reviews and analyses have been completed.

(3) Low-level requirements are documented, reviewed, and traceable to high-level requirements.

(4) The source code implements and is traceable to the low-level requirements and has been reviewed.

b. ***Data Required for the Software Development Review.*** For a software development review, the software data shown in Table 4-2 should be made available to the certification authorities. The supporting software data should be under configuration control, as appropriate for the software level, prior to the review. The data listed in table 4-1 should also be available during the development review.

Software Data	**ED-12B / DO-178B Section**
*Software Requirements, Design and Code Standards	11.6, 11.7, 11.8
Software Requirements Data	11.9
Design Description	11.10
Source Code	11.11
Software Verification Procedures (as applied to ED-12B / DO-178B, Annex A, Tables A-2 through A-5)	6.3.1, 6.3.2, 6.3.3, 6.3.4, 11.13
Software Verification Results (as applied to ED-12B / DO-178B, Annex A, Tables A-2 through A-5)	6.3.1, 6.3.2, 6.3.3, 6.3.4, 11.14
Software Life Cycle Environment Configuration Index	11.15
Problem Reports	11.17
Software Configuration Management	11.18
Software Quality Assurance Records (as applied to ED-12B / DO-178B, Annex A, Tables A-2 through A-6)	11.19

* Not required for Level D, per ED-12B / DO-178B, Annex A, Table A-1.

Table 4-2 Data Availability for the Software Development Review

c. ***Evaluation Criteria for the Software Development Review.*** The objectives which apply to development in ED-12B / DO-178B, Annex A, Tables A-2 (objectives 1-6), A-3 (all objectives), A-4 (all objectives), A-5 (objectives 1-6), A-8 (objectives 1-4, 6), A-9 (objectives 1-2), and A-10 (objective 3), should be used as evaluation criteria for this review. Additionally, the software life cycle data should be evaluated to determine the effectiveness of the applicant’s implementation of the plans and standards in the development process.

4.5.3 Software Verification Review

a. ***Identification of the Software Verification Review Process.*** The software verification process is typically a combination of inspections, demonstrations, reviews, analyses, tests, and test coverage analysis. As with the other reviews, the software configuration management and quality assurance processes are also active during these verification activities. The verification activities confirm that the software product that was specified is the software product that was built. The software verification review should, therefore, ensure that the software verification processes will provide this confirmation and will result in objective evidence that the product has been sufficiently tested and is the intended product. The purpose of the software verification review is to: assess the effectiveness and implementation of the applicant's verification plans and procedures; ensure the completion of all associated software configuration management and quality assurance tasks; ensure that the software requirements, design and the integration of the code have been verified; and ensure that the software verification process will achieve the requirement-based test coverage and structural coverage criteria of ED-12B / DO-178B, Annex A, Table A-7. Before conducting a software verification review, the software verification process should be sufficiently complete and mature to ensure that representative verification data exists to assess that the applicant’s approved plans and standards are being complied with and evidence exists that transition criteria have been met. The following criteria are indicative of a mature verification process:

(1) All development data items (e.g., requirements, design, source code, object code, linking and loading data, executable image) are complete, have been reviewed, and are under configuration control.

(2) Test cases and procedures are documented, reviewed, and placed under configuration control.

(3) Test cases and procedures have been executed (either formally or informally).

(4) Completed test results are documented, as agreed to in the planning documents.

(5) The software testing environment is documented and controlled.

b. ***Data Required for the Software Verification Review.*** For the purpose of compliance findings for the software verification review, the software data shown in Table 4-3 should be made available to the certification authorities. The supporting software data should be under configuration control, as appropriate for the software level, prior to the review. The data listed in tables 4-1 and 4-2 should also be available during the verification review.

Software Data	**ED-12B / DO-178B Section**
Software Requirements Data	11.9
Design Description	11.10
Source Code	11.11
Software Verification Cases and Procedures	6.3.1-6.3.6, 11.13
Software Verification Results	11.14
Software Life Cycle Environment Configuration Index (test environment)	11.15
Software Configuration Index (test baseline)	11.16
Problem Reports	11.17
Software Configuration Management Records	11.18
Software Quality Assurance Records	11.19
Software Tool Qualification Data	12.2.3

Table 4-3 Data Availability for Software Verification Review

c. ***Evaluation Criteria for Software Verification Review.*** The following ED-12B / DO-178B, Annex A, objectives apply to the software verification review and should be used as the evaluation criteria for the review: Tables A-1 (objective 3), A-5 (objective 7), A-6 (all objectives), A-7 (all objectives), A-8 (all objectives), A-9 (objectives 1-2), and A-10 (objective 3).

4.5.4 Final Certification Software Review

a. ***Identification of Final Certification Software Review.*** The final software build establishes the configuration of the software product considered by the applicant to comply with all the objectives of ED-12B / DO-178B. It is the version of the software that is intended to be used in the airborne application. The purpose of this review is to: determine compliance of the final software product with the objectives of ED-12B / DO-178B, as defined by the software level and other software policy and guidance; ensure that all software development, verification, quality assurance, configuration management, and certification liaison activities are complete; ensure a software conformity review has been completed and the software complies; and review the final Software Configuration Index documents (SCIs) and the Software Accomplishment

Summary (SAS). The final certification software review should take place when the software project is completed and includes the following criteria:

(1) The Software conformity review has been performed and any deficiencies resolved.

(2) The Software Accomplishment Summary and Software Configuration Indexes have been completed and reviewed.

(3) All software life cycle data items have been completed, approved, and placed under configuration control.

b. ***Data Required for Final Certification Software Review.*** For the purpose of this review, all the software life cycle data items of ED-12B / DO-178B should be available to the certification authorities. However, only the data shown in Table 4-4 is of special interest for this review. The supporting software data should be under configuration control, appropriate for the software level, prior to the review.

Software Data	**ED-12B / DO-178B Section**
Software Verification Results	11.14
Software Life Cycle Environment Configuration Index	11.15
Software Configuration index	11.16
Software Configuration Management Records	11.18
Problem Reports	11.17
Software Quality Assurance Records (including the Software Conformity Review Report)	11.18
Software Accomplishment Summary	11.20

Table 4-4 Data Availability for Final Certification Software Review

c. ***Evaluation Criteria for Final Certification Software Review.*** Evaluation criteria for this review include all the objectives of ED-12B / DO-178B, Annex A. Additionally, all software-related problem reports, action items, certification issues, etc. must be addressed prior to certification or authorisation.

4.5.5 Summary

The following table provides an overview of the information presented in the preceding sub-sections in relation with the scope of the various software reviews and audits.

Audit N°	Major Reviewed Devices	Documentation available during the audit	Entry Criteria
1 Software Planning Review	Planning. SW partitioning. Safety objectives. Software tools / tools policy. QA policy.	System requirements. PSAC*, SDP*, SVP*, SCMP*, SQAP*. Software requirements, design and code standards. TQP*. Life cycle data of qualified tools. Verification, configuration management and process assurance records.	As soon as possible.
2 Software Design Review	Software Requirements vs. System requirements (traceability). Software design vs. design standards. code vs. standards. Verification Plan vs. Software requirements and design. Design verification activity. Follow-up of the previously open actions.	Software requirements data. Software design data. All life cycle data down to code. Verification, configuration management and process assurance records. All data previously mentioned.	When at least 75% of the design life cycle data is available at maintained in configuration.
3 Software Verification Review	Software requirements coverage (correctness and robustness). Follow-up of the previously open actions.	Software Verification Cases and Procedures Software Verification Results Verification, configuration management and process assurance records Problem reports All data previously mentioned	When at least 75% of the verification data is available at maintained in configuration.
4 Final Software Certification Review	Coverage of tests (integration / validation) Traceability of the final documentation package Traceability of change request / Problem Reports Status of open actions. Supplier quality actions	Evolution/Problem reports. Records. SAS*. SCI*. TAS for development/verification tools. Verification, configuration management and process assurance records.	Once all SW activities are finished and at least 1 month prior to final system / equipment certification review.

* To be submitted to the authorities at least ten working days before the audit

4.6 ADDITIONAL CONSIDERATIONS FOR THE SOFTWARE REVIEW PROCESS

a. Although this section of the Certification Memorandum proposes four types of on-site reviews, the type, number, and extent of those reviews may not be suitable for every certification project and applicant. Additional considerations and alternative approaches may be appropriate. The following list of considerations may influence the level of involvement of the certification authorities in the software review process:

(1) The software level (s), as determined by a system safety assessment.

(2) The product attributes (e.g. size, complexity, system functionality, software design).

(3) The use of new technologies or unusual design features.

(4) Proposals for novel software methods or life cycle model(s).

(5) The knowledge and previous success of the applicant in software development to comply with the objectives of ED-12B / DO-178B.

(6) The availability, experience, and authorisation of staff responsible for conducting software certification-related activities.

(7) The existence of issues associated with Section 12 of ED-12B / DO-178B in the project.

(8) The issuance of CRIs for software-specific aspects of the certification project.

b. On-site software reviews may be increased or decreased in number. Four reviews is a typical number for a Level A or Level B project. Fewer or no reviews may be appropriate for some equipment manufacturers. Furthermore, reviews may be merged into a combined review. It is the responsibility of the certification authority representative to determine the desired level of investigation, to plan the reviews, and to co-ordinate with the applicant.

4.7 PREPARING, CONDUCTING, AND DOCUMENTING A SOFTWARE REVIEW

This paragraph of the Certification Memorandum provides guidelines for preparing for an on-site review, conducting an on-site review, and recording and communicating the results of the review:

a. **Prepare for the On-Site Review.** The responsible certification engineer should assemble the review team. The team should include at least one person knowledgeable in software engineering, one person familiar with the type of system being evaluated, and a manufacturing inspector knowledgeable in software quality assurance and configuration management (if available). The certification engineer should co-ordinate with the applicant regarding the upcoming software review at least six weeks in advance and propose an agenda. To optimise the efficiency of the review team while on-site, the certification authorities should request the applicant to send each team member the software plans identified in ED-12B / DO-178B, Section 4.3, 15 working days prior to the review (if not agreed differently between EASA and the applicant). Each team member should review the plans prior to arriving at the applicant's facility. The certification engineer should prepare a short entry briefing to introduce the team members, restate the purpose of the review, and review the agenda. The applicant should provide a short briefing to facilitate an understanding of the system under review, the software life-cycle model, processes, tools used, and any additional considerations.

b. **Notify the Applicant.** The responsible certification authority representative should notify the applicant in writing regarding the certification authorities' expectations in the software review. The following information should be included in the notification letter:

(1) The purpose of the review and the type of review (i.e., planning, development, verification, or final).

(2) The date and duration of the review.

(3) A list of certification authority review participants with contact information.

(4) A request that the software plans identified in ED-12B / DO-178B, Section 4.3, be sent to each review participant.

(5) A request that all pertinent life cycle data should be made available at time of review.

(6) An indication of which ED-12B / DO-178B objectives will be assessed.

(7) A suggestion that the applicant should conduct their own self-assessment prior to the review.

(8) A request that the responsible managers, developers, verification, configuration management, and quality assurance personnel be available for questions.

c. **Conduct the On-site Review.** A typical on-site review includes the following elements:

(1) Certification Authority Entry Briefing to Include: introduction of review team members; restatement of purpose of the review; and overview of the review agenda.

(2) Software Developer's Briefing to Include: availability of facilities; availability of life cycle data; personnel schedule constraints; overview of the system; interaction of the system with other systems; system architecture; software architecture; software life cycle model (including tools and methods); progress against previous action items or CRIs (if appropriate); current status of the development; and any additional considerations (per ED-12B / DO-178B, Section 12).

(3) Certification authorities' review of the applicant/developer's processes.

(4) Certification authorities' review of the product.

d. **Record the Review Results.** The review results should be recorded; the records should include the following, as a minimum:

(1) A list of the each life cycle data item reviewed to include: document name; control identity; version and date; requirement identification (where applicable); source code module (where applicable); paragraph number (where applicable); and review results.

(2) The approach taken to establish the finding or observation.

(3) An explanation of the findings or observations as related to the objectives of ED-12B / DO-178B (documented with detailed notes). Each unsatisfied objective requires a summary of what was done and a discussion as to why the objective was not satisfied. Examples should be included, when necessary. This will ensure that the approach and findings can be understood and reconstructed at some future date.

(4) Any necessary actions for either the applicant or the certification authorities.

(5) Listing of all current or potential CRIs.

e. **Deliver an Exit Briefing.** The final briefing to the applicant and / or the developer under review should be factual and positive and should summarise the findings and observations from the review. Findings and observations should be presented with specific reference to ED-12B / DO-178B, the certification basis, policy, guidance, or other certification documentation. The applicant and/or developer should be given the opportunity to respond to the findings and observations.

f. **Prepare a Review Report.** During the review, the applicant should produce a review report to summarize all the review findings, observations, and required actions. The report should be reviewed and agreed with the certification authority representative and the developer before the end of the review.

g. **Identify and Prepare CRIs (as needed).** CRIs are a means of documenting technical and certification issues that must be resolved prior to system certification. They provide the necessary communication between applicant and certification engineer and management. CRIs should be identified, prepared, and resolved as soon as possible after the issue is discovered. Co-ordination with the PCM and/or EASA should be established, as dictated by the applicable project procedures.

5 ORGANISATION, ROLE AND LEVEL OF INVOLVEMENT OF EASA AND APPLICANTS IN SOFTWARE PROJECTS

5.1 PURPOSE

The purpose of this section is to present (for information) the role of the EASA software panel and the EASA SW/CEH group, how they determine their level of involvement in a certification project and to describe their relations with the other EASA system panels.

In a similar manner to the one described in this section, an applicant may choose to tailor the software review process (described above in section 4) with respect to specific criteria such as software DAL, complexity, supplier experience, the presence of novelties etc., so as to determine their own level of involvement in the software of each system. This tailoring may be performed at the company level or at the product level.

When an applicant has determined their own level of involvement for the software of each system, the applicant should produce a document for EASA concurrence that lists the CSCIs in all the systems on the aircraft and shows the DAL and the applicant's planned level of involvement for each CSCI.

NOTE: In addition to this section, the description of the EASA organisation, its role and level of involvement in each specific software project may be extended in the Project Information Document (PID) where it is applicable.

5.2 BACKGROUND

a. Modern aircraft and engine designs include many items of integrated digital equipment, some of which perform critical functions. The certification activities of the software panel need to be well organised and closely coordinated with the activities of each system panel. The system panels involved include:

   Panel 1: Flight

   Panel 2: Performance

   Panel 3: Structures

   Panel 4: Hydro Mechanical systems

   Panel 5: Electrical systems

   Panel 6: Avionics systems

   Panel 7: Powerplant and fuel systems

   Panel 8.1: Cabin Safety

   Panel 8.2: Environmental Control systems

   Panel 12: Safety

   The system/software integration of many types of equipment brings the need for close coordination between system and software specialists. Each panel that is in charge of a system expected to use digital equipment shall be the primary panel for the certification requirements relevant to that system. The software panel stands as the secondary panel for some of those requirements (mainly chapters 1301 and 1309 from Certification Specifications or CS-E 50 (d,f)). The SW/CEH experts will perform the verification activities for the software documents under their responsibility and will issue recommendations for compliance statements to the software panel as well as to the relevant system panels.

b. EASA relies on the applicant's DOA system so they can be confident that compliance with the certification requirements applicable to the software has been achieved.

   Within the scope described in Subpart J of Part 21 (particularly cf. § 21A.239, 21A.257 and 21A.263), the applicant shall be entitled to perform design activities and to propose documents for acceptance without further verification by EASA.

   The level of involvement of the EASA SW/CEH group for each piece of equipment in a given project can vary between the levels of NONE, LOW, MEDIUM or HIGH involvement in the certification activities. Additionally, depending on the stage of advancement and the quality of the certification activities already performed, the level of involvement initially agreed by EASA for a given piece of equipment may evolve during the project.

5.3 DISCUSSION

5.3.1 Organisation and role of the SW/CEH group

a. Coordination within the SW/CEH group (when applicable)

(1)The coordinator

The software coordinator is in charge of coordination of the software aspects of certification for the program on behalf of the EASA team. He is a member of the software panel.

i. Within the SW/CEH group, the coordinator

- is the focal point in the case where no member has been designated for the software aspects of certification for some on-board equipment and in this case, he/she may propose another SW/CEH group member to be designated
- is the focal point in the case of the relevant SW/CEH expert being unavailable due to conflicting priorities and in this case, he/she may propose an alternate SW/CEH expert to be designated
- is the focal point within the EASA team for resolving generic issues linked to software development/certification policy.

ii. In addition, the coordinator should be informed:

- by the applicant and/or by the relevant EASA systems specialist or SW/CEH expert of new developments affecting the certification of the software installed on the aircraft (except for minor changes)
- periodically (at least twice a year) by the applicant of the overall software certification activities scheduled and should ensure that all SW/CEH group experts are adequately allocated in order to carry out the associated software certification activities in due time.

iii. Finally, the coordinator should report to the PCM :

- periodically (at least twice a year) the results of the overall software certification activities carried out and attend relevant status meetings (e.g. Type Board Meetings)
- on PCM request, any relevant software certification activity findings made by the SW/CEH group.

(2)Work distribution among the SW/CEH group

The **SW/CEH panel** is responsible for the definition and acceptance of the software certification basis and the acceptable means of compliance.

The **SW/CEH group** is responsible for the acceptance that the software development process is in line with the certification basis (including methodologies for software development) and consistent with the DAL allocated by the relevant system panel.

b. Coordination with system panels

(1) Determination of the Certification basis and the Acceptable Means of Compliance.

The relevant SW/CEH group member should be invited to the familiarisation meetings for systems that embed software features for which the SW/CEH group will have to assess compliance. The SW/CEH group member should assist the applicant and the relevant system panel in the determination of the certification basis. This task includes the definition of the applicable requirements and interpretative material as well as the identification of the generic software CRIs that are applicable to the system.

In addition, the designated SW/CEH group member may recommend the system specialist and SW/CEH panel to open a new CRI and may submit proposals. The draft CRI will then be further developed by the SW/CEH panel with support from the relevant SW/CEH group member and the relevant system panel if needed. The endorsement of the SW/CEH panel is necessary to issue the EASA position on this issued CRI.

(2) Development Assurance Level (DAL) allocation

Acceptance of the DAL allocation at system level is the responsibility of the system specialist, based on the Functional Hazard Analysis (FHA) or the Preliminary System Safety Analysis (PSSA). In order to assess the DAL allocation proposed by the applicant, the system specialist may request advice from the relevant SW/CEH group member.

This SW/CEH group member is responsible for assessing the DAL allocation within software components, provided this allocation remains consistent with the system DAL allocation.

For this purpose, the applicant or the system panel should provide the SW/CEH group with the system FHA and any document justifying the DAL allocation, including a DAL downgrading justification (if applicable).

(3) Compliance statement

The SW/CEH group member is responsible for the compliance verification activities that he/she performs: at the end of the compliance verification, he/she shall issue a compliance statement to the PCM and send a copy of it to the system panel that is the primary panel for the system with a copy to the SW/CEH co-ordinator and applicant.

The SW/CEH panel co-ordinator is responsible for issuing the final software panel compliance statement. As the primary panel, the system panel is responsible for the final compliance statement. If there is any inconsistency between the system panel compliance statement and the software compliance statement (for example, where a system panel issues a compliance statement even though some of the corresponding software documents have not received a compliance statement recommendation from the SW/CEH group), the issue shall be brought up and solved at PCM level.

5.3.2 Determination of EASA software level of involvement (LOI)

a. General

The software certification process involves both the EASA software and CEH experts and the applicant's DOA system.

Early coordination should take place between the EASA SW/CEH group and the applicant during an initial certification meeting in order to specifically address their involvement in the software certification activities.

The agenda and objectives of the initial certification meeting should cover the following topics:

(1) The applicant should present to EASA the characteristics of the software as well as the organisational context of the software development and certification (including the identification of the suppliers).

(2) The applicant should present to EASA the activities they plan to monitor (a list of reviews and a schedule) and state the rationale for the activities they plan to conduct under their DOA system.

(3) The EASA SW/CEH group should present their intended overall level of involvement.

(4) The EASA SW/CEH group should present their audit planning and define the documentation to be delivered before each audit.

Agreement on the level of involvement, audit planning and documentation to be submitted to EASA should be ideally reached during the initial certification meeting. The final decision and a reference to the rationale will be documented in the system certification plan.

b. Determination of the LOI

The outcome of the assessment performed during the initial certification meeting will result in a level of involvement of NONE, LOW, MEDIUM or HIGH for the EASA SW/CEH group in the certification activities. There are five major criteria that can influence the outcome of this assessment:

(1) The Software Criticality Level

(2) The complexity of the software development

(3) The software certification experience of the development team and/or applicant

(4) The service history of the software

(5) The need for a new EASA policy due to any novelties (such as new technology, new design methods, unusual tools, etc.)

5.3.3 Influence of the LOI on the certification activities

a. EASA Software audits

Section 4 of this Certification Memorandum provides information regarding the software review process. Depending on the level of involvement agreed, the number of software audits can be adapted as described in table 5-1:

LOI	Software audits
HIGH	At least 2 on-site audits (e.g. SOI#1 + combined SOI#2 and SOI#3) + desktop reviews (e.g. SOI#4) + additional technical meetings (e.g. novelty) + Review of applicant Review Reports (cf. b.)
MEDIUM	At least 1 on-site audit (e.g. combined SOI#2 and SOI#3) + desktop reviews (e.g. SOI#1+ SOI#4) + additional technical meetings + Review of applicant Review Reports (cf. b.)

LOI	Software audits
LOW	1 on-site audit or desktop reviews + Review of applicant Review Reports (cf. b.)
NONE	*Eventual review of applicant Review Reports* (cf. NOTE)

Table 5-1 Software audits depending on LOI

NOTE: In particular cases, EASA can increase their involvement.

b. Applicant Software Review Reports

For each development that is not completely performed under the applicant's DOA system, a status report will be presented by the staff member responsible for certification during each software audit, in particular covering the software review activities performed by the applicant and the Action Items in progress.

The applicant should report to EASA about their own monitoring as follows:

(1) Software Review Reports will be sent for information to the EASA SW/CEH group until the Entry Into Service (EIS) of the product.

(2) Software Review Reports should be made available to the EASA SW/CEH group at least 15 working days (if not agreed differently between EASA and the applicant) before a planned EASA software audit, or 15 working days before the target date for EASA software approval. In any case, Software Review Reports shall be available for consultation during EASA audits.

c. Documentation to be submitted to EASA

The SW/CEH group experts and the system specialists should agree with the applicant early in the project on the categories of documents they wish to review or receive for information.

The applicant will send software certification documents to the SW/CEH group and send system certification documents to the relevant system panels. In addition, some system documents may be sent to the SW/CEH group for information only (e.g. the FHA), and some software documents may be sent to system panels (e.g. SAS) for information only. The allocation of certification documentation between the system panel and the SW/CEH group shall be clearly documented in the system certification plans.

The table 5-2 gives an example of the documents that fall under the responsibility of the SW/CEH group, depending on the LOI:

	Certification Documents to be provided				
LOI	**PSAC**	**SAS**	**CID**	**Other SW plans**	**Software Review Reports**
HIGH	For agreement	For agreement	For information	For information	For information
MEDIUM	For agreement	For agreement	For information	For information	For information
LOW	For information (in cases of no EASA involvement)	On request	On request	On request	For information
NONE	On request	On request	On request	On request	On request

Table 5-2 Documentation to be provided depending on LOI

NOTE: the classification "for agreement", "for information" and "on request" can be classified in categories, e.g.:

CAT1 = "for agreement"

CAT2 = "for information"

CAT3 = "on request".

5.3.4 Revision of LOI

At any time, the level of involvement initially agreed between EASA and the applicant for a given item of equipment may be revised. It can evolve either towards more involvement or towards less involvement, depending on the stage of advancement and the quality of the certification activities already performed.

6 RESERVED

7 GUIDELINES FOR THE APPROVAL OF FIELD LOADABLE SOFTWARE (FLS)

7.1 PURPOSE

This section provides guidelines for approving field-loadable software (FLS). These guidelines are applicable to software data approvals related to type certification, supplemental type certification and approvals of changes to type design.

7.2 BACKGROUND

Through technological advances, the field loading of software has become a common process. This process reduces aircraft down-time for maintenance and increases the efficiency of maintaining airborne equipment. ED-12B / DO-178B, Section 2.5, provides some system design considerations for FLS; however, the existing guidance for the approval of FLS through the processes for TCs, STCs and the approval of changes to type design is limited. This section provides additional guidelines for the approval of FLS using the processes for TCs, STCs or approvals of changes to type design.

7.3 THE USE OF EARLIER VERSIONS OF ED-12

Versions of ED-12 earlier than revision B do not provide any guidance regarding FLS, and should not be used as a means of compliance for FLS approvals. For software developed to previous guidelines, at least the field-loadable component and the protective schemes of the component should be demonstrated to meet the guidelines contained in ED-12B / DO-178B or an alternate means of compliance, as agreed to between the applicant and the certification authorities.

7.4 APPROVAL OF FIELD-LOADABLE SOFTWARE (FLS)

The following procedures should be implemented as part of the processes for TCs, STCs or approvals of changes to type design for the approval of FLS:

a. It should be confirmed that the software meets the objectives of ED-12B / DO-178B or another acceptable means of compliance, as agreed to between the applicant and the certification authorities.

b. It should be confirmed that the considerations outlined in ED-12B / DO-178B, Section 2.5, have been addressed.

c. It should be confirmed that the software and hardware configuration were verified/tested together during the verification process (i.e., the software must be installed on the target computer in which the approval was granted).

d. There should be a Configuration Management process in place to assure that the installation configuration (i.e., software part number, the hardware part number, the aircraft/ engine model, and the aircraft/engine serial number combination, as applicable) is the same configuration that was approved during the TC, STC or approvals of changes to type design process.

e. If redundant parts on the aircraft or engine are field-loadable, the applicant should define the following: (1) the requirements for intermixing different software loads on the parts, (2) requirements for partially successful and partially unsuccessful loads, and (3) the aircraft or engine dispatch ability effects of successful and unsuccessful loads on redundant parts.

f. There should be a process in place to assure that the software loaded is the software approved and that the software has not been corrupted (e.g., verification with an appropriate data transfer integrity check, such as a Cyclic Redundancy Check (CRC)).

NOTE 1: A "product" is an aircraft, an engine or a propeller.

NOTE 2: Different CRC algorithms give different assurances that the data transferred is correct. The applicant and approving authority should assure that the algorithm used is sufficient for the integrity required for the software level of the data being loaded.

g. If there is no process in place to assure that paragraph 7.4f above has been addressed, the applicant should plan and provide alternative means to systematically verify the proper loading of the FLS after each loading (e.g. by means of an installation test procedure). It should be ensured that the on-board loading system is approved considering the following items:

   (1) The applicant should demonstrate that the on-board loading system complies with the guidelines of ED-12B / DO-178B, Section 2.5 or an alternate means of compliance as described in paragraph 7.4 of this Certification Memorandum.

   (2) The applicant should provide documentation defining the operation of the on-board loading system and the recommended means for maintaining configuration control of equipment by the operator. This documentation should include guidelines for the configuration control processes that meet the guidelines outlined in this Certification Memorandum.

   (3) The applicant's on-board loading system and procedures should be approved by the certification authorities. Depending on the implementation, this approval may include the data loader, as well as the procedures. (Note: Many approaches to data loading do not require evaluation of the data loader because integrity checks are built into the data and the data transfer process, see paragraph 7.4 f of this Certification Memorandum).

   (4) If the applicant proposes more than one medium for on-board loading (e.g. diskette, mass storage, etc.), loading from all mediums should comply with the guidelines in this section of the Certification Memorandum.

h. The applicant should demonstrate the ability to verify the airborne equipment software part number with on-board equipment, carry-on equipment, or other appropriate means.

i. It should be confirmed that any changes to FLS will undergo a software change impact analysis to determine the safety impact and the classification of the changes as major or minor.

j. FLS which is also user-modifiable and has been approved by the certification authorities as user-modifiable does not require further determinations of compliance for dissemination and installation (reference ED-12B / DO-178B, Section 2.4).

7.5 INSTALLATION CONSIDERATIONS

The approved FLS may be installed on the aircraft via a Service Bulletin, Engineering Change Request, or another means approved by the certification authorities. The approved means of installation may vary, depending upon the method for granting approval. Whether the FLS approval is through TC, STC, approvals of changes to type design or some other approval process, the document used to install the FLS should be approved by the certification authorities or under DOA privileges. The document should specify the following elements:

a. The aircraft/engine and hardware applicability.

b. Verification procedures to assure that the software was correctly loaded into an approved and compatible target computer.

c. Any post load verification and/or test procedures required to show compliance with the guidelines specified in this section of the Certification Memorandum.

d. Actions to be taken in the event of an unsuccessful load (for example, to prohibit the dispatch of the aircraft).

e. Reference to an approved loading procedure.

f. Maintenance record entry procedures required to maintain configuration control.

g. Reference to the Aircraft Flight Manual or Operations Manual of the aircraft, as appropriate.

7.6 MAINTENANCE AND PART MARKING CONSIDERATIONS

Maintenance and part marking for FLS should be performed in accordance with Part 21 and the appropriate parts of the Certification Specifications. Additional maintenance and part marking considerations that apply specifically to FLS using TC, STC or approvals of changes to type design processes are discussed below:

a. The applicant's Aircraft Maintenance Manual or Instructions for Continued Airworthiness (ICA) should include the procedures to be followed when conducting maintenance on airborne equipment using field-loadable software.

b. The applicant's Aircraft Maintenance Manual or IFCA should include a procedure that requires maintenance personnel to verify the software part number configuration before and after maintenance is performed on the airborne equipment.

   NOTE: If the software loading cannot be verified (e.g., procedures do not render proper results, checksum fails, part number does not match approved part number, etc.), the system should not be considered functional and the aircraft should not be dispatched.

c. It is the responsibility of maintenance personnel to ensure the FLS part number is recorded in the necessary maintenance logs.

d. For airborne equipment having separate part numbers for hardware and software, the software part numbers need not be displayed on the outside of the unit, as long as they can be verified through some kind of electronic query. It is the maintenance personnel's responsibility to ensure that the software part number has been logged. When new software is loaded into the unit, the same requirement applies and the approved software part number should be verified before the unit is returned to service.

e. For airborne equipment having only one part number, which represents a specific configuration of software and hardware, the unit identification on the nameplate should be changed when the new software is loaded. When new software is loaded, the software part number stored in the target computer after data loading should be verified electronically. It should be verified that the electronic software part number and the unit part number displayed on the nameplate are an approved configuration prior to returning the unit to service.

f. Changes to software part number, version, and/or operational characteristics should be reflected in the Operations Manual, Aircraft Flight Manual, IFCA, and/or any other appropriate document.

8 RESERVED

9 GUIDELINES FOR THE APPROVAL OF AIRBORNE SYSTEMS AND EQUIPMENT CONTAINING USER-MODIFIABLE SOFTWARE

9.1 PURPOSE

This section provides guidelines regarding the application of ED-12B / DO-178B for approval of airborne systems and equipment designed to contain user-modifiable software components. These guidelines are applicable to the approval of airborne systems and equipment and the software aspects of those systems related to type certificates (TC), supplemental type certificates (STC) and approvals of changes to type.

9.2 SCOPE

This section applies to user-modifiable software only. The guidance provided below applies neither to option-selectable software nor to field-loadable software, except where such software is also user-modifiable. For guidance on field-loadable software see section 7 of this Certification Memorandum.

9.3 THE USE OF EARLIER VERSIONS OF ED-12 / DO-178

Versions of ED-12 / DO-178 prior to version B did not provide any guidance regarding user-modifiable software, and should not be used as a means of compliance for user-modifiable software approvals. For software developed to previous guidelines, at least the user-modifiable component, the protective schemes, and any affected aspects of the non-modifiable component should be developed to ED-12B / DO-178B or another acceptable equivalent means as agreed to between the applicant and the certification authorities. ED-12B / DO-178B guidance for user-modifiable software is contained in Sections 2.4, 5.2.3, 7.2, 11.1, and 11.10 of that document. ED-12B / DO-178B also provides guidance for upgrading software from previous guidance in Section 12.1.4.

9.4 SAFETY CONSIDERATIONS

a. User-modifiable software is software within an airborne system approved for user modification. Users (e.g., airlines, operators) may modify user-modifiable software within the specified modification constraints and with approved modification procedures without any further involvement by the certification authorities. It is intended that once the system with the user-modifiable software has been certified, the certification authorities should require no further visibility, review, or approval of modifications made to that user-modifiable software component. Therefore, modification of the user-modifiable software by the user should have no effect on the aircraft safety margins or operational capabilities, flight crew workload, any non-modifiable software components, or any protection mechanisms of the system.

b. A user-modifiable software component is that part of the software within the airborne system that is designed and intended to be changed by the user. A non-modifiable software component is one that is not designed or intended to be changed by the user. Modification constraints for UMS should be developed by the applicant and provided to the users. Any change that affects safety margins, operational capabilities, flight crew workload, any non-modifiable software components, protection mechanisms, or software boundaries, or that results in exceeding a pre-approved range of data, parameters, or equipment performance characteristics warrants the rescinding of the classification of the software as user-modifiable, and requires design approval under the applicable regulations.

   <u>NOTE:</u> Multiple trim values used as user-modifiable software that may affect safety

require special attention. In general, it is not acceptable to simply test the trim value throughout its trim range, because of the uncertainty for acceptability of all the combinations of the trims. In most cases, it is not possible to verify all possible combinations of multiple trims. Therefore, in the case of multiple trims used as user-modifiable software, acceptance of verified sets of trims is generally required.

c. The potential effects of user-modifiable software modification must be determined by the system safety assessment and mitigated by system and software design means, development and verification assurance, approved procedures, and approved tools (if applicable). When evaluating data as part of the ED-12B / DO-178B process, the applicant and the certification authorities representatives involved should ensure that the protective mechanisms, verification, and user-modification procedures do not interfere with the non-modifiable components or with protection integrity. The applicant should obtain the concurrence of the certification authorities early in the program as to the acceptability of the protective mechanism, protection verification, and modification procedures and tools.

   NOTE: The purpose of the protective mechanism is to ensure that the user-modifiable component does not interfere with the non user-modifiable component. This protective mechanism should be evaluated during the initial approval of the system that contains user-modifiable software. It should be assured that no modification of the software by the user affects the protective mechanism. Paragraphs 9.7 and 9.8 of this section will further address protection.

9.5 CONSIDERATIONS FOR DISPLAYED DATA

Where information is displayed to the flight crew and is derived from UMS, the information should be identified to distinguish it as "advisory data only" that has not been approved as part of the aircraft and/or engine type design by the certification authority. If the information displayed has received an operational approval as part of the operational procedures of the aircraft and/or engine by an appropriate operation approval authority, this distinction may not be necessary. If the design or inherent nature of the equipment or user-modifiable component makes the distinction between approved and unapproved information so readily apparent to the flight crew that errors distinguishing the two types of information are reasonably precluded, explicit identification of the information as "advisory data only" may not be required. Such identification, where required, should be provided by a non-modifiable component and allow the flight crew to readily distinguish between information approved or accepted by the certification or operational approval authority. "Advisory data only" information should be verifiable by the flight crew from another source on the aircraft and/or engine, should not be used to display any information where the potential worst case failure condition for displaying misleading data is any greater than minor, or should not be used by the flight crew in performing any aircraft and/or engine operational procedures (for example, supplemental situational awareness only).

9.6 MODIFICATION OF AIRCRAFT AND/OR ENGINE PERFORMANCE PARAMETERS

Modifications that could affect the safety margins, operational capabilities of the aircraft and/or the engine, or crew workload include modifications of displayed data or other data used by the flight crew to determine aircraft and/or engine performance parameters. These types of modifications require certification authority approval. Modification of the user-modifiable component to provide or revise these parameters, regardless of whether they are provided as primary or advisory information, requires certification authority approval. Such a change would warrant rescinding the classification of the software as user-modifiable and would require design approval and part number revision.

9.7 PROTECTION

Non-modifiable software components of the airborne system should be protected from user-modifiable software components. The system requirements should specify the protection mechanisms that prevent the user modification from affecting system safety, operational capability, or flight crew workload. If the system requirements do not include provisions for user modification, the software should not be modified by the user. The protection mechanism should be assigned the assurance level associated with the most severe failure condition of the system as determined by the system safety assessment. If software provides the protection mechanism for user-modifiable software, that software protection should be assigned the highest software level of the system as determined by a system safety assessment. The protection should prevent any modification or failure of the user-modifiable software from causing a loss of protection. Protection integrity cannot depend on any activities being accomplished by the user. The protection integrity should be such that it can neither be breached accidentally or intentionally. The applicant-provided means of modification of the user-modifiable software should be shown to be the only means by which the modifiable component can be changed.

9.8 TOOLS USED TO PROTECT NON-MODIFIABLE COMPONENTS

a. ED-12B / DO-178B, Section 5.2.3, requires that the non-modifiable software components be protected from modifiable components in order to prevent interference with the safe operation of the non-modifiable software components. To enforce this protection, the use of the tools to make the changes to the modifiable component is allowed. If such tools will be used to enforce this protection, then the following information should be provided to the certification authorities for approval:

   (1) plans for controlling the tool version;

   (2) plans for controlling tool usage;

   (3) plans for qualifying or verifying the tool; and

   (4) procedures for performing modifications to the tool.

b. Software forming a component of the tool and used in the protective function should be developed to the software level associated with the most severe failure condition of the system, as determined by a system safety assessment.

c. The use of software tools as (part of) the protective function requires tool qualification and approval of procedures to use and maintain the tool. Changes to the tool or procedures may require re-qualification of the tool.

9.9 DATA REQUIREMENTS

a. The applicant should identify in the Plan for Software Aspects of Certification (PSAC) their intention to develop an airborne system that will contain a user-modifiable software component(s). The PSAC should also describe the means of complying with ED-12B / DO-178B (including the design considerations of ED-12B / DO-178B Section 5.2.3), the protection mechanism, and the means of ensuring the integrity of the protection mechanisms. If software tools will be used for the modification, the PSAC should also identify tool qualification plans or verification procedures to ensure that the tool has modified the user-modifiable software to approved procedures and constraints and has not affected the non-modifiable software or protection mechanisms.

b. The Software Development Plan and the software design data should specify the design methods and details of implementation for ensuring protection from user modifications.

c. The Software Configuration Index should identify the approved procedures, methods, and tools for making modifications to the user-modifiable software, including tool qualification data, if applicable. There is no need to update the Software Configuration Index of equipment containing user modifiable software if the changes are limited to the user modifiable component, and made within the constraints established during the original certification project.

d. The Software Accomplishment Summary should summarise the entire development and verification of the non-modifiable software components, user-modifiable software component(s), protection mechanism, and modification procedures and tools, including tool qualification, if applicable. There is no need to update the Software Accomplishment Summary of equipment containing user modifiable software if the changes are limited to the user modifiable component, and made within the constraints established during the original certification project.

9.10 OTHER CONSIDERATIONS

At the time of the user modification, the user assumes responsibility for all aspects of the user-modifiable software components and tools used for modifying the software, including software configuration management, software quality assurance, and software verification. User modifications should be performed according to approved procedures established by the system requirements and software data, using approved tools. If the user makes any modification to the non-modifiable software components, the protection mechanisms, the approved procedures, or the approved tools, other than those established by the system requirements and approved procedures, then they have violated the type design, and the type certificate of the aircraft and/or engine may be rescinded.

NOTE: A system to track or log software modifications that fall under the description in this section should be considered where appropriate so that both the Certification and Continued Airworthiness aspects of the modifications may be reviewed by the cognizant authorities, as needed.

10 GUIDELINES FOR APPLYING THE ED-12B / DO-178B LEVEL D CRITERIA TO PREVIOUSLY DEVELOPED SOFTWARE (PDS)

10.1 PURPOSE

This section provides guidelines regarding the application of ED-12B / DO-178B to previously developed software (PDS) that has been categorised as contributing to no more than a Minor failure condition on the aircraft. AMC 20-115B recognises ED-12B / DO-178B as a means to secure EASA certification of digital computer software. ED-12B / DO-178B assigns a software level of D to any software that can cause or contribute to no more than a Minor aircraft failure condition. However, the application of the objectives associated with Level D software are frequently misinterpreted, especially when applied to software that was not originally approved using ED-12B / DO-178B (i.e., PDS). This section should be used to apply ED-12B / DO-178B to PDS that is categorised as Level D.

10.2 BACKGROUND

ED-12B / DO-178B provides for five different levels of software based on the contribution of the software to potential failure conditions. These software levels represent differing levels of development process rigor based on the severity of the potential failure conditions that the software can cause or to which the software can contribute. Level D is assigned to software that can cause or contribute to no more than a Minor aircraft failure condition.

ED-12B / DO-178B contains 28 objectives for Level D software that should be satisfied before certification is granted.

To be consistent with a Minor aircraft failure condition, the intent of Level D software objectives is to provide a thorough investigation of the functional behaviour of the software and to provide the necessary configuration control. However, some of the required objectives for Level D have been misinterpreted when considered with the overall objective of establishing correct functional behaviour.

Due to confusion over Level D objectives, the application of ED-12B / DO-178B for these systems has not been consistent over different projects. Many developers may decide to do more than the stated requirements for Level D; however, this section of the Certification Memorandum concentrates on the minimum requirements. Proper application of Level D objectives permits the use of PDS, which is software that was not originally approved using ED-12B / DO-178B (e.g., Commercial-off-the-shelf (COTS) software, software developed using military standards, software developed using ED-12 / DO-178 or ED-12A / DO-178A, software developed using other industry standards). Refer to Section 12.1 of ED-12B / DO-178B for additional guidance for using PDS. In particular, Section 12.1.4 should be referenced for additional considerations when upgrading a previous development baseline. While this section addresses PDS, these guidelines may also be applicable for other software that is required to meet the ED-12B / DO-178B Level D objectives.

Section 12 of this Certification Memorandum contains guidance for the certification of software changes to legacy systems.

10.3 DISCUSSION

A consistent interpretation of ED-12B / DO-178B for Level D software is important for the certification of PDS software. Of the 28 objectives found in ED-12B / DO-178B for Level D software, experience has shown that there are five objectives that are frequently misinterpreted. One of the objectives is related to integral processes; the remaining four objectives are related to source code, software architecture, and low level requirement definitions. The discussion presented in this section is applicable to any ED-12B / DO-178B Level D software certification. Paragraph 10.4 of this Certification Memorandum provides specific procedures for the certification of Level D PDS.

a. Objective 1 in ED-12B / DO-178B, Annex A, Table A-1, "Software development and integral processes activities are defined." A number of field-experience comments point to the absence of any requirement for Level D software to comply with Objective 6 in ED-12B / DO-178B, Annex A, Table A-1 which states "Software Plans comply with this document (i.e., ED-12B / DO-178B)" and have concluded that there should not be a requirement to comply with Objective 1 which states "Software development and integral processes activities are defined." However, Objective 1 ensures that even for Level D software: (1) there are some plans (e.g., Plan for Software Aspects of Certification, Software Development Plan, Software Configuration Management Plan, Software Quality Assurance Plan, Software Verification Plan), even if the plans themselves do not comply with ED-12B / DO-178B, and (2) those plans are followed (see Objective 1 in ED-12B / DO-178B, Annex A, Table A-9). Additionally, the plans should enable compliance with the ED-12B / DO-178B objectives applicable for Level D software.

b. Objective 4 in ED-12B / DO-178B, Annex A, Table A-2, "Low-level requirements are developed." For Level D software, the intent of this objective is to assure that the low-level requirements are defined. However, Table A-4 objectives related to the architecture and low-level requirements require no explicit verification of the software architecture and low-level requirements for Level D software, except for verifying the integrity of any software partitioning. Therefore, Objective 4 of Table A-2 is satisfied implicitly by satisfying Objectives 1 and 2 in ED-12B / DO-178B, Annex A, Table A-6. The satisfaction of Objectives 1 and 2 demonstrate that the executable object code complies with and is robust with high-level requirements. Since there is no objective for Level D to ensure that the executable code is compatible with the low-level requirements, it is not necessary to ensure for Level D software that the low-level requirements are traceable to the high-level requirements.

c. Objective 3 in ED-12B / DO-178B, Annex A, Table A-2, "Software architecture is developed." The logic as applied in paragraph 10.3(b) above may be applied to Objective 3 (i.e., Objective 3 is implicitly satisfied by other objectives and does not need to be explicitly satisfied for Level D PDS, since Table A-4, Objectives 8 through 12, do not require verification of the software architecture).

d. Objective 5 in ED-12B / DO-178B, Annex A, Table A-2, "Derived low-level requirements are defined." The referenced Section in ED-12B / DO-178B for Objective 5 (i.e., Section 5.2.1b) states that "Derived low-level requirements are provided to the system safety assessment process," rather than just "defined." As with the low-level requirements and software architecture, there is no explicit verification of derived low-level requirements for Level D software. The satisfaction of this objective is implied by satisfying Objective 2 in ED-12B / DO-178B, Annex A, Table A-2, "Derived high-level requirements are defined" and the associated verification of high-level requirements.

e. Objective 6 in ED-12B / DO-178B Annex A, Table A-2, "Source code is developed." The actual ED-12B / DO-178B referenced text for Objective 6 (i.e., Section 5.3.1a) states, "Source code is developed that is traceable, verifiable, consistent, and correctly implements low-level requirements." However, according to Annex A, Table A-5, there are no verification objectives for Level D source code. Therefore, there is no requirement to establish consistency between source code, low-level requirements, and high-level requirements. The consistency requirement is between the executable code and the high-level requirements for Level D. The objective is for the executable code to meet all of the functional requirements. Furthermore, the existence of object code implies the existence of source code so that Objective 6 of ED-12B / DO-178B, Annex A, Table A-2 is reasonably covered by satisfying other objectives (i.e., Objectives 1 and 2 of Table A-2; Objective 2 of Table A-3; Objectives 1 and 2 of Table A-6; and Objective 3 of Table A-7) for level D software.

10.4 PROCEDURES

For a project involving the approval of Level D PDS, the procedures listed below should be followed either by the cognizant certification authority specialist or included as appropriate in DOA procedures:

a. Software reviewers should review the software plans to assure that: (1) some plans exist (e.g., Plan for Software Aspects of Certification, Software Development Plan, Software Configuration Management Plan, Software Quality Assurance Plan, Software Verification Plan); (2) those plans are followed (reference ED-12B / DO-178B, Annex A, Table A-9, Objective 1); and (3) the plans enable compliance with ED-12B / DO-178B objectives for Level D software.

b. Software reviewers can ensure that low-level requirements, software architecture, derived low-level requirements, and source code are defined and exist for Level D software; however, software reviewers should not assess the quality or compliance of these artefacts with ED-12B / DO-178B objectives and software life cycle data content requirements, except where necessary to ensure that software partitioning integrity is confirmed (Objective 13 of Table A-4). The intent for Level D of these objectives will be satisfied by the objectives for Level D for Tables A-6 and A-7.

c. When evaluating the PDS, the following steps should be followed:

   (1) The applicant should verify that a failure condition or malfunction of the Level D software can cause or contribute to no more than a Minor failure condition.

   (2) The applicant should identify the functions to be used from the PDS, and any software developed to specifically mitigate any failures or malfunctions of the PDS (for example, wrapper code, partitioning, or monitors). The certification authority should confirm that safety implications are addressed.

   (3) The applicant should ensure that the PDS cannot result in any unacceptable failure condition in the target application.

d. In cases where there are multiple software levels within a given system and/or component, the protection and associated mechanisms between the different software levels (such as partitioning, safety monitoring, or watchdog timers) should be verified to meet the objectives of the highest level of software associated with the system component. This can occur when there are multiple functions in a component (e.g., maintenance and navigation) or when there are different categorisations of types of failure conditions, such as loss of function versus a corrupted function (e.g., misleading display data). An example of the latter case is a navigation system supported by a PDS operating system. The loss of the navigation function can be shown to produce only a Minor aircraft failure condition, whereas misleading navigation is usually considered to be a Major aircraft failure condition. If the navigation function is protected (partitioned) from the operating system in such a way that any failure of the operating system can be shown to produce only a loss of function, then the operating system only needs to be evaluated to Level D criteria. However, the applicant needs to verify that the operating system can really only contribute to a loss of navigation function and not to a misleading navigation failure condition. The applicant also needs to verify that common-cause and common-mode losses of identical functions or common resources cannot result in a worse failure condition than was originally assigned to the individual system. In this case, part of the development effort would be to demonstrate that the PDS can be shown to meet all the Level D objectives, as outlined above.

e. It is theoretically possible for Level D software to operate in conjunction with software of other levels. In this case a thorough protection/partitioning analysis should be performed in conjunction with the system safety assessment. However, discussion of protection/ partitioning is outside the scope of this section and will not be further discussed.

f. See ED-12B / DO-178B, Section 12.1, for additional guidance on the use of PDS.

11 GUIDELINES FOR THE QUALIFICATION OF SOFTWARE TOOLS USING ED-12B / DO-178B

11.1 PURPOSE

This section provides guidelines regarding the application of ED-12B / DO-178B to the qualification of software verification and development tools. AMC 20-115B recognises ED-12B / DO-178B as a means to secure EASA certification of software in airborne systems and equipment. Section 12.2 of ED- 12B addresses tool qualification; however, the Section 12.2 criteria are often misinterpreted, resulting in inconsistent application in the field. This section of the Certification Memorandum clarifies the application of ED-12B / DO-178B in the area of tool qualification but does not change the intent of ED-12B / DO-178B in this area. The guidelines in this section of the Certification Memorandum should be used in applying the criteria in ED-12B / DO-178B for the qualification of tools.

11.2 BACKGROUND

Section 12.2 of ED-12B / DO-178B states that qualification of a tool is needed when processes in ED-12B / DO-178B "are eliminated, reduced, or automated by the use of a software tool, without its output being verified as specified in Section 6" of ED-12B / DO-178B. ED-12B / DO-178B states, "The objective of the tool qualification process is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced, or automated." The items below provide further information regarding tool qualification:

a. Software development can be a very repetitive and human-labour intensive process. This can result in errors, as well as high costs. For these reasons, various tools have been developed to automate portions of this process. If the tools are dependable, then improvements in productivity and lower numbers of in-service errors may be realised.

b. In order to certify systems developed with tool support, both certification authorities and applicants need to obtain confidence by qualification that these tools are dependable. ED-12B / DO-178B Section 12.2 was designed to provide criteria for establishing which tools require additional confidence and the criteria and data needed to establish that confidence. However, several provisions of this Section of ED-12B / DO-178B are difficult to interpret. This section of the Certification Memorandum clarifies the intent of ED-12B / DO-178B Section 12.2 and its application.

c. Some areas that have resulted in misinterpretation and inconsistent application of the ED-12B / DO-178B tool qualification criteria are:

    (1) When a tool should be qualified.

    (2) Justification for the different criteria for qualifying software development tools and software verification tools.

    (3) Which criteria apply to software development tools and which apply to software verification tools.

    (4) Data to be produced for software development tools and for software verification tools.

    (5) Acceptance criteria for tool operational requirements.

    (6) Tool determinism.

    (7) Tool partitioning assurance and evidence.

    (8) Tool configuration control.

d. These areas have resulted in inconsistencies in applying the criteria within ED-12B

Section 12.2 to certification projects. This section of the Certification Memorandum is designed to address the above problems by clarifying the intent and application of ED-12B Section 12.2.

11.3 DISCUSSION

a. Not all software tools require qualification. According to ED-12B / DO-178B Section 12.2, qualification of a tool is needed only when processes described in ED-12B / DO-178B are eliminated, reduced, or automated by the use of that tool without its output being verified as specified in ED-12B / DO-178B Section 6. This means that if the results of the tool are being relied on to supply the sole evidence that one or more objectives are satisfied, the tool must be qualified per ED-12B / DO-178B Section 12.2. If the output of the tool is verified by some other means, then there is no need to qualify the tool. For example, if all the outputs of a test case generator are reviewed to ensure that coverage is achieved, then the tool does not need to be qualified.

b. ED-12B / DO-178B Section 12.2 identifies two types of tools: software verification tools and software development tools. Each type will be discussed below.

c. ED-12B / DO-178B defines verification tools as "tools that cannot introduce errors, but may fail to detect them."

(1) The following are examples of verification tools:

i. A tool that automates the comparison of various software products (e.g., code, design) against some standard(s) for that product.

ii. A tool that generates test procedures and cases from the requirements.

iii. A tool that automatically runs the tests and determines pass/fail status.

iv. A tool that tracks the test processes and reports whether or not the desired structural coverage has been achieved.

(2) Many claim that verification tools can be more reliable than humans in a number of verification tasks, if their correct operation is demonstrated. In order to encourage the use of verification tools, ED-12B / DO-178B Section 12.2 was designed to provide an acceptable approach to qualifying verification tools.

d. ED-12B / DO-178B defines development tools as “tools whose output is part of airborne software and thus can introduce errors.” If there is a possibility that a tool could generate an error in the airborne software that would not be detected, then the tool cannot be treated as a verification tool. An example of this would be a tool that instrumented the code for testing and then removed the instrumentation code after the tests were completed. If there was no further verification of the tool’s output, then this tool could have altered the original code in some unknown way. Typically, the original code before the instrumentation was introduced is what is used in the product. This example is included to demonstrate that tools used during verification are not necessarily verification tools. The effect on the final product must be assessed to determine the classification of the tool.

e. The reason for the distinction between development and verification tools is based on the likelihood of allowing an error into the airborne system. For development tools there is a potential to introduce errors directly into a system. However, a verification tool can only fail to detect an error that already exists in the product. Tools, therefore, need to be deficient in two different processes to allow an error to get into the airborne software: the development process introducing the error and the verification process failing to detect the error. For this reason, ED-12B / DO-178B calls for different levels of rigor in the qualification of verification and development tools.

11.4 PROCEDURES

For any project involving the qualification of tools, certification authority representatives should follow the procedures and guidelines listed in this sub-paragraph of the Certification Memorandum:

a. Guidelines for determining whether a tool should be qualified:

(1) Whether a tool needs to be qualified is independent of the type of the tool (development or verification). There are three questions to ask to determine whether a tool needs qualification. If the answer is "Yes" to all of the questions below, the tool should be qualified:

i. Can the tool insert an error into the airborne software or fail to detect an existing error in the software within the scope of its intended usage?

ii. Will the output of the tool not be verified or confirmed by other verification activities, as specified in Section 6 of ED-12B / DO-178B?

iii. Are any processes of ED-12B / DO-178B eliminated, reduced, or automated by the use of the tool? That is, will the output from the tool be used to either meet an objective or replace an objective of ED-12B / DO-178B, Annex A?

(2) Once it has been determined that a tool does not require qualification, the remainder of ED-12B / DO-178B Section 12.2 is not applicable to that tool. In order to ensure a timely response, the certification authority should be involved early in the tool qualification agreements for the certification project.

(3) The Plan for Software Aspects of Certification (PSAC) should include a listing of all software tools and justification for why each tool does or does not require qualification.

NOTE: The inclusion of all software tools in the PSAC is encouraged to provide early visibility of tools that may require qualification.

b. Guidelines for determining which tool qualification criteria apply to development tools and which criteria apply to verification tools:

(1) Table 11-1 applies to tools requiring qualification and can be used to determine which criteria of ED-12B / DO-178B Section 12.2 apply to which type of tool. Table 11-1 shows the similarities and differences in the qualification criteria for development and verification tools. The column in Table 11-1 titled "Criteria" summarises the ED-12B / DO-178B requirement; the column titled "Dev./Ref." lists the applicability of the criteria for development tools and the appropriate ED-12B / DO-178B Section reference; and the column titled "Verif./Ref." lists the applicability of the criteria for verification tools with the appropriate ED-12B / DO-178B Section reference.

Criteria	**Dev./Ref.**	**Verif./Ref.**
Only deterministic tools may be qualified (to be further clarified in paragraph 11.4f of this Certification Memorandum).	Yes/12.2	Yes/12.2
Qualification should only be for a specific system; the intention should be stated in the PSAC.	Yes/12.2	Yes/12.2
Combined tools should be qualified to ED-12B / DO-178B, Section 12.2.1 unless partitioning can be shown (to be further clarified in paragraph 11.4g of this Certification Memorandum).	Yes/12.2.b	Yes/12.2.b
Software configuration management and software quality assurance process objectives should be applied to tools being qualified (to be further discussed in paragraph 11.4h of this Certification Memorandum).	Yes/12.2.c	Yes/12.2.c
Qualification should satisfy the same objectives as the airborne software.	Yes/12.2.1.a	No
The software level of the tool may be reduced.	Yes/12.2.1.b	No
A trial period may be used as a means to demonstrate compliance with the tool operational requirements.	Yes/ 12.2.1.c	Yes/12.2.2
Tool Operational Requirements should be reviewed.	Yes/12.2.1.d(1)	Yes/12.2.2
Compliance with Tool Operational Requirements under normal operating conditions should be demonstrated.	Yes/12.2.1.d(2)	Yes/12.2.2
Compliance with Tool Operational Requirements under abnormal operating conditions should be demonstrated.	Yes/12.2.1.d(3)	No
Requirements-based coverage should be analysed.	Yes/12.2.1.d(4)	No
Structural coverage appropriate for the tool’s software level should be completed.	Yes/12.2.1.d(5)	No
Robustness testing appropriate for the tool’s software level should be completed.	Yes/12.2.1.d(6)	No
Potential errors should be analysed.	Yes/12.2.1.d(7)	No

Table 11-1 ED-12B / DO-178B Criteria Applicable to Tool Qualification

c. Guidelines for data submittal and data availability to demonstrate tool qualification. The guidelines for data to support tool qualification are listed throughout ED-12B / DO-178B Section 12.2; however, there is no definitive guidance as to the minimum level/amount of data to be submitted to the certification authorities for tool qualification. The data submittals vary according to the type of tool being developed. Even though there are some similar guidelines for the two tool types, the data requirements for each tool type are different. Table 11-2 summarises the required tool qualification data. The column titled “Data” lists the required data for tool qualification. The column titled “Applicability” summarises whether the data is applicable for development tool qualification (Development) or verification tool qualification (Verification). The column titled “Available/Submit” summarises whether the data should be submitted to the certification authorities or just available for certification authority review. The column titled “ED-12B / DO-178B Ref.” lists the ED-12B / DO-178B Section(s) that reference the criteria. The remainder of this paragraph discusses the tool qualification data summarised in Table 11-2.

Data	Applicability	Available/Submit	ED-12B / DO-178B Ref.
Plan for Software Aspects of Certification (PSAC)	Verification & Development (see Note 1 below)	Submit	12.2, 12.2.3.a, & 12.2.4
Tool Qualification Plan	Development Only (see Note 2 below)	Submit	12.2.3.a(1), 12.2.3.1, & 12.2.4
Tool Operational Requirements	Verification & Development	Available	12.2.3.c(2) & 12.2.3.2
Software Accomplishment Summary (SAS)	Verification & Development (see Note 1 below)	Submit	12.2.4
Tool Accomplishment Summary	Development Only (see Note 2 below)	Submit	12.2.3.c(3) & 12.2.4
Tool Verification Records (for example, test cases, procedures, and results)	Verification & Development	Available	12.2.3
Tool Qualification Development data (e.g., for example, design, and code)	Development	Only Available	12.2.3

Table 11-2 Data Required for Tool Qualification

NOTE 1: For development tool qualification, the PSAC should reference the Tool Qualification Plan and the SAS should reference the Tool Accomplishment Summary.

NOTE 2: The Tool Qualification Plan and the Tool Accomplishment Summary may be developed for verification tool qualification, if the applicant so desires.

(1) Verification Tool Qualification Data. Of the two tool qualification types, verification tools require the fewest data submittals and availability. Data for verification tool qualification are discussed below:

i. For verification tools, the applicant should specify the intent to use a verification tool in the PSAC (reference ED-12B / DO-178B, Section 12.2). The PSAC should be submitted to the certification authorities and should include the intended tool qualification schedule. This alerts the certification authority representative to provide a response to the intended use of the tool and opens a dialogue on acceptable qualification methods and documentation approaches. The certification authority should provide a written response to the applicant on the acceptability of the approach listed or referenced in the PSAC in a timely manner (i.e., the verification tool qualification approaches in the PSAC should be reviewed and approved or addressed in a timely manner).

ii. For verification tool qualification, the Tool Operational Requirements should be documented and made available to the certification authorities (see ED-12B / DO-178B, Section 12.2.3.2). The requirements for the Tool Operational Requirements data are discussed in paragraph 11.4d of this Certification Memorandum.

iii. Data showing that all of the requirements in the Tool Operational Requirements have been verified should also be documented and available for certification authority review. Sufficient verification data is needed to demonstrate normal operation only and will vary depending on the complexity of the tool, the purpose of the tool, and how the tool is used. The applicant may package these verification data in any document they choose.

iv. An entry summarising the results of the verification tool qualification should be included in the Software Accomplishment Summary (SAS). The SAS should be submitted to the certification authorities. This allows the certification authority to approve the results of the verification data and provides evidence of the tool qualification status.

NOTE: The applicant may choose to provide a separate Tool Qualification Plan and Tool Accomplishment Summary referenced by entries in the PSAC and the SAS for software verification tools. Entries are still required in the PSAC and SAS. This is an acceptable approach and has the added benefit of providing the ability to reference a data package for reuse in subsequent certifications or in different certifications where the usage of the tool can be shown to be identical.

(2) Development Tool Qualification Data. There are additional requirements for a software development tool. The criteria for qualifying a software development tool are similar to the approval process for the airborne software. For software development tool qualification, the following data submittal and availability items should be considered:

i. The actual qualification approach and data to be provided are specified in the Tool Qualification Plan. The Tool Qualification Plan should be submitted to the certification authorities for approval.

ii. The Tool Accomplishment Summary should also be submitted to the certification authorities for approval. It summarises the results of the tool qualification process and describes and references the relevant tool qualification data.

iii. The PSAC and SAS should be submitted to and approved by the certification authorities. However, these documents will likely only reference the Tool Qualification Plan and the Tool Accomplishment Summary documents.

iv. The Tool Operational Requirements should be documented and made available to the certification authorities (reference ED-12B / DO-178B, Section 12.2.3.2). The requirements for the Tool Operational Requirements data are discussed in paragraph 11.4d of this Certification Memorandum.

v. Data that shows that all of the requirements in the Tool Operational Requirements have been verified should also be documented and made available for certification authority review. Sufficient verification data is needed to demonstrate under normal and abnormal operational conditions. The data will vary depending on the complexity of the tool, the purpose of the tool, and how the tool is used. The applicant can package this verification data in any document they choose.

vi. Other tool qualification data, such as design, code, test cases and procedures, etc. should be available for certification authority review.

(3) The certification authority representatives involved should strive to use the document format and media used by the applicant. Any repackaging for submittal to the certification authorities should be undertaken only when the certification authorities are unable to review the data in the manner presented by the applicant or if the applicant is unable to meet the data retention requirements.

d. Guidelines for evaluating acceptability of Tool Operational Requirements data: Tool Operational Requirements for any tool that requires qualification should be completed and made available for certification authority review. A complete set of operational requirements is necessary to communicate to both the user and the certification authority what the tool does, how it is used, and the environment in which it performs. The Tool Operational Requirements should identify all functional and technical features of the tool and the environment in which it is installed (see ED-12B / DO-178B, Section 12.2.3.2). The information required is different depending on the type of tool:

(1) For a verification tool, the Tool Operational Requirements should provide at least the following information:

i. The tool's functionality in terms of specific requirements that are verified as part of the tool's qualification tests.

ii. A definition of the tool's operational environment, including operating system and any other considerations (e.g., an analysis of what tools will not do and what is required to cover that shortage (e.g., extensions to checklists, test cases) and any specialised hardware requirements (e.g., processors, special test equipment, or interfaces).

iii. Any other information necessary for the tool's installation or operation (e.g., User's Manual) should be included in the Tool Operational Requirements.

(2) A development tool needs to include all the information listed above for verification tools but should also include at least the following:

i. Software development processes performed by the tool.

ii. Expected response under abnormal operating conditions.

NOTE: In some cases the User’s Manual or other supplier documentation may contain the needed information. Where additional information is included over and above the required information, the required information should be clearly identified. In the case where there is insufficient information from the tool supplier, the applicant should provide the missing information.

e. <u>Guidelines on acceptable verification of the Tool Operational Requirements:</u> Development and verification tools require verification of the Tool Operational Requirements. For verification tools, only verification over the normal operating conditions is required; whereas for development tools, verification over the abnormal operating conditions is also required. ED-12B / DO-178B Sections 6.4.2.1 and 6.4.2.2 describe verification for normal and abnormal conditions and will not be covered in this section of the Certification Memorandum. However, since the operational requirements may contain additional information not directly related to the verification activity (e.g., the appearance of menus, dialog boxes, configuration), additional guidance is needed to reduce unnecessary verification for verification tools. For verification tools only, those portions of the operational requirements that are used directly in the setting up, conducting, monitoring, and reporting of verification need to be verified as part of tool qualification. The applicant should ensure that those features/portions of the verification tool that are not used have no adverse impact on those features/portions that are being used. If additional features are used at a later time, then additional verification will be required.

f. <u>Guidelines on the interpretation of the determinism of tools:</u>

(1) Although only deterministic tools can be qualified (see section 12.2 of ED-12B / DO-178B), the interpretation of determinism is often too restrictive. A restrictive interpretation is that the same apparent input necessarily leads to exactly the same output. However, a more accurate interpretation of determinism for tools is that the ability to determine correctness of the output from the tool is established. If it can be shown that all possible variations of the output from some given input are correct under any appropriate verification of that output, then the tool should be considered deterministic for the purposes of tool qualification. This results in a bounded problem.

(2) This interpretation of determinism should apply to all tools whose output may vary beyond the control of the user, but where that variation does not adversely affect the intended use (e.g., the functionality) of the output and the case for the correctness of

the output is presented. However, this interpretation of determinism does not apply to tools that have an effect on the final executable image embedded into the airborne system. The generation of the final executable image should meet the restrictive interpretation of determinism.

(3) As an example, a tool may have a graphical user interface that allows the user to interact in a diagrammatic fashion. Underlying this tool are data tables that capture the intended meaning of those diagrams. Often, however, the output from these tools is at least partially driven by the physical ordering of the entries in these data tables, and the ordering of the data table entries is not controlled by the tool user. However, the correctness of the tool's output can be established. With the restrictive interpretation of determinism, this tool could not be qualified. However, with the expanded interpretation, qualification may be possible.

g. Guidelines for qualifying combined development and verification tools:

(1) The guidelines in this paragraph apply only to tools which provide combined development and verification functions where the output of both the development and the verification functions are being used to eliminate, reduce, or automate processes of ED-12B / DO-178B. Combined tools that are used to eliminate, reduce, or automate only development objective(s) or only verification objective(s) should be qualified as such irrespective of the other capabilities present in that tool.

(2) Qualification of combined tools (when both the development and verification functions are being used to meet or replace objectives of ED-12B / DO-178B) should be performed to the guidance equivalent to the airborne software level *unless protection/partitioning between the two functions can be demonstrated*. Acceptable evidence of this protection/partitioning would be to show that the output of one function of the tool has no effect on the output of the other function of the tool (i.e., the tool capabilities are functionally isolated).

(3) When protection/partitioning between the development and verification functions is shown, the protected/partitioned functions may be qualified as if they were separate development and verification tools (i.e., the verification functions may be qualified to the criteria for verification tools).

h. Guidelines on configuration management of qualified tools: In order to receive credit (i.e., meet or replace ED-12B / DO-178B objectives) for the use of qualified tools, those tools must be kept under configuration management. Not all of the requirements for configuration management of tools are contained in ED-12B / DO-178B Section 12.2. Section 12.2.3b of RTCA/DO-178B specifies the control categories for development and verification tool qualification data (see also Section 7.2.9b of RTCA/DO-178B). The control category for development tool qualification data should be the same as that required for airborne software of the same level (that is, the "CC1" and "CC2" criteria in Annex A tables applies to development tool qualification data). Verification tool qualification data, on the other hand, may be categorized as control category #2.

i. Guidelines on verifying changes to previously qualified tools: A software change impact analysis should be conducted on all changes to tools that have been previously qualified. The analysis should be thorough enough to assess the impact of the tool change on the product, as well as other tools under the influence of the change. A regression analysis may form part of the change impact analysis.

j. Guidelines for tools developed before AMC 20-115B issuance: Software tools used on pre-ED-12B / DO-178B projects may be qualified for use on projects where ED-12B / DO-178B is the means of compliance, if they meet the guidelines of this section. As an alternative, service history may be considered for such tools (see Section 4.11 of RTCA/DO-248B for more information on qualification of tools using service history).

12 GUIDELINES FOR THE CERTIFICATION OF SOFTWARE CHANGES IN LEGACY SYSTEMS USING ED-12B / DO-178B

12.1 PURPOSE

This section provides guidelines regarding the application of ED-12B / DO-178B to software for systems that were developed prior the issuance of JAA Temporary Guidance Leaflet No. 4: "Recognition of EUROCAE ED-12B / RTCA DO-178B", on 1 October 1996. These systems are referred to as legacy systems throughout this section and refer to systems developed under ED-12 or ED-12A.

AMC 20-115B, published on Nov 5th, 2003 (EASA Decision 2003/12/RM) recognises ED-12B / DO-178B as an acceptable means of compliance for the evaluation of software in airborne systems. The ED-12B / DO-178B guidance for legacy systems is frequently misinterpreted and is not being consistently applied. This section of the Certification Memorandum does not change the intent of ED-12B / DO-178B with regard to legacy systems but clarifies the application of ED-12B / DO-178B.

12.2 BACKGROUND

On January 11, 1993, the FAA issued AC 20-115B which recognised DO-178B as a means to secure FAA certification of digital computer software. Prior to the issuance of AC 20-115B, many airborne systems were approved using DO-178 or DO-178A. These systems are referred to as legacy systems.

On 1 October 1996, the JAA published Temporary Guidance Leaflet (TGL) No. 4, which also recognised ED-12B / DO-178B as a means to secure JAA certification of digital computer software. Systems developed under ED-12 or ED-12A thus also became legacy systems.

TGL N°4 was superseded by ACJ 20-115B (published in GAI 20 dated 1st May 2002).

ACJ 20-115B was superseded by AMC 20-115B, published on Nov 5th, 2003 (EASA Decision 2003/12/RM).

Many manufacturers are striving to use ED-12B / DO-178B on their legacy systems. There are several items to keep in mind when addressing the use of ED-12B / DO-178B on legacy systems:

a. ED-12B / DO-178B is different from the two previous versions of ED-12. The major change from the previous versions is the emphasis on a set of co-ordinated objectives rather than a collection of unrelated goal statements. There is also a change from an emphasis on documentation to an emphasis on process objectives and the data needed to demonstrate compliance with those objectives. Software testing is the most visible difference between ED-12B / DO-178B and previous versions. Therefore, software in legacy systems approved under a previous version may not have the same level of software testing assurance as that invoked by ED-12B / DO-178B (i.e., ED-12B / DO-178B clarifies the scope and extent of software testing and test coverage). For new projects, AMC 20-115B effectively cancels all previous versions of ED-12 / DO-178 as acceptable means of compliance in new projects. Therefore, changes/modifications to software accepted prior to the issuance of AMC 20-115B should be evaluated using ED-12B / DO-178B when they are migrated to new aircraft and/or engines.

b. Another difference between ED-12B / DO-178B and earlier versions is the classification of software levels and the need to perform a safety assessment to determine the software level. Previous versions only recognised three software levels, whereas ED-12B / DO-178B recognises five software levels. ED-12B / DO-178B provides no guidance to show correspondence between these levels. This section of the Certification Memorandum will provide a method to establish that correspondence. Once the correspondence has been established, then guidance provided by ED-12B / DO-178B may be applied to upgrade from a lower level to a higher level.

c. Prior versions of ED-12 do not address the qualification of software development and verification tools. In many cases tools are involved in making changes to legacy systems. Therefore, modification projects for legacy systems are faced with the issue of how to address tools that were used and not evaluated as part of the original certification. The subject of tool qualification is specifically addressed in section 11 of this document. Section 11.4 j addresses pre ED-12B / DO-178B tools.

d. After reviewing field experience with numerous changes, a procedure was developed to provide a more consistent approach to address changes to the software of legacy systems. The approach described in this section takes advantage of previous certifications while ensuring that software changes are properly implemented and satisfy current regulations and guidance. (Note: If the system contains multiple levels of software, the procedure should be applied to each of the partitioned sections that is affected by the change).

12.3 DISCUSSION

a. If the software level of the legacy system cannot be shown to be equivalent to or better than that required by the product installation being considered, then the software should be upgraded in accordance with procedures defined in ED-12B / DO-178B Section 12.1.4, "Upgrading a Development Baseline." This will require a complete re-evaluation to demonstrate assurance to the appropriate objectives of ED-12B / DO-178B. Determining equivalence is addressed in paragraph 12.4 of this Certification Memorandum; however, application of ED-12B / DO-178B Section 12.1.4 is not addressed further in this Certification Memorandum.

b. There are four variables that can affect the actions needed in response to changes to software in legacy systems:

   (1) the certification basis for the original product or installation of the legacy system containing the legacy software, (i.e. the regulations, the RTCA/DO-178 version, and software level applied to the original approval),

   (2) whether ED-12B / DO-178B or a previous version is the accepted means of compliance for software for the product or installation under consideration, (and whether the software level is the same as or equivalent to the software level for the original certification),

   (3) whether the software is being modified or unchanged, (and how many other times it has been changed since the original certification, and the reason for those changes), and

   (4) whether the software and the legacy system are being installed on the same or a different aircraft and/or engine.

c. Assuming that the software levels can be shown to be equivalent, the majority of legacy system issues of concern can be categorised into the following groups:

   (1) Legacy systems software is not modified and is reinstalled on the original aircraft and/or engine (see paragraph 12.4b of this Certification Memorandum)

   (2) Legacy systems software is not modified but is installed on a different aircraft and/or engine where ED-12B / DO-178B is not adopted as the means of compliance for software (see paragraph 12.4b of this Certification Memorandum).

   (3) Legacy systems software is modified and is reinstalled on the original aircraft and/or engine (see paragraph 12.4c of this Certification Memorandum).

   (4) Legacy systems software is modified and is installed on a different aircraft and/or

engine where ED-12B / DO-178B is not adopted as the means of compliance for software (see paragraph 12.4c of this Certification Memorandum).

(5) Legacy systems software is modified and is installed on a different aircraft and/or engine where ED-12B / DO-178B is adopted as the means of compliance for software (see paragraph 12.4d of this Certification Memorandum).

(6) Legacy systems software is not modified but is installed on a different aircraft and/or engine where ED-12B / DO-178B is adopted as the means of compliance for software (see paragraph 12.4e of this Certification Memorandum).

d. Legacy systems, by definition, already have a recognised approval for installation or manufacturing through the Type Certificate (TC), Supplemental Type Certificate (STC) or Approvals of Changes to Type Design processes. If there are no changes to the software of these systems, then the original approval of the software may still be valid, assuming that equivalence to the required software level for the current installation can be ascertained (as further discussed in paragraph 12.4 of this Certification Memorandum). Prior to installation of the system in an aircraft or engine, there should be an assessment to ensure that the legacy system will not be used in a significantly different manner than that covered by the original installation certification.

e. Systems with small, simple changes should be handled as changes under the original certification basis (i.e., ED-12B / DO-178B does not need to be applied to the changes).

Examples of software changes that might be classified as small and simple include:

- Gain changes where the new gain is within a band of gain settings originally tested,
- Changes to maintenance information formatting,
- Adding an additional output interface,
- Changing data in a personality module that is within the set of options previously verified and approved.

1) The certification authorities should be able to readily establish that these changes have been performed correctly under the original certification basis. The normal data submittals appropriate to the revision of ED-12 / DO-178 used for the original certification will still need to be evaluated to ensure that the changes are implemented correctly. If this cannot be done, then this is not a small, simple change.

2) The determination of whether a change is small, simple cannot be made by objective considerations such as metrics or a count of lines of code. Therefore, this determination will be based on the individual judgement of the certification authorities specialists involved.

NOTE: This process of allowing small, simple changes should not be followed, if the system is being used differently than the original certification project, or if the system has experience service difficulties.

f. When changes are made to legacy systems beyond the small, simple changes, assurance that the changes have been correctly implemented and verified will be required. The following items should be considered:

(1) Earlier versions of ED-12 / DO-178 do not contain well-defined acceptance criteria for several of the objectives and guidelines. One example is in the area of testing. ED-12B / DO-178B requires that testing be of sufficient rigour to provide specific structural coverage criteria and provide specific criteria for that rigour, whereas ED-12A / DO-178A only requires that testing exercise the logic and computations but does not specify any criteria for how extensively the logic should be exercised.

(2) Additionally, some newer technologies and tool qualification are not even addressed in the earlier versions of ED-12 / DO-178. In all cases where ambiguities exist, the material in ED-12B / DO-178B should be used to provide a more exact interpretation.

(3) To be consistent with prior certifications, ED-12B / DO-178B should be used to evaluate the processes used to make the change, the changed software components, and those components affected by the software changes. Affected components should be identified by performing a change impact analysis of the software changes and identifying impacts on other components, interfaces, timing, memory, etc. (e.g., control coupling analysis, data coupling analysis, timing analysis, memory usage analysis, etc.). These analyses should also identify the level and extent of regression testing needed to verify the change.

(4) The unaffected portions of the software already have a certification basis and could be accepted in accordance with paragraph 12.3c of this Certification Memorandum. (It should be noted that the unaffected portion is the software that neither changed nor was affected by the change as determined by analysis of control flow, data flow, or timing. The change impact analysis is used to determine the affected and unaffected portions.)
In most cases, the risk of latent errors remaining in the software may be further mitigated by considering the benefit of service experience in conjunction with the prior certification. ED-12B / DO-178B Section 12.3.5, "Service Experience," contains a number of criteria that should be satisfied to allow the use of service experience. By virtue of the previous certification of the software, it may be assumed as already meeting many of the provisions of ED-12B / DO-178B Section 12.3.5. Little or no additional data may be needed from the applicant regarding service experience under Section 12.3.5 if the applicant has sufficient relevant service history data and no in-service problems have occurred with the system.

Note: The note in Section 12.3.5g of ED-12B / DO-178B does imply that additional data may be required to verify system safety objectives for software components and should be appropriately considered.

(5) Once a DO-178B compliant change process is in place to address a major software change, that process should be applied to all subsequent changes to that software.

12.4 PROCEDURES

For any project involving changes to a legacy system or a different installation for a legacy system, the certification authorities should follow the procedures listed in this section of the Certification Memorandum.

a. The certification authorities should establish that there is equivalence between the legacy system's software level (s) and the proposed installation's software level using Table 12-1 below. Table 12-1 illustrates the equivalence between ED-12/DO-178, ED-12A/DO-178A and ED-12B / DO-178B. Table 12-1 is designed as a truth table asking the following question: "If the Legacy System has a specific ED-12/ED-12A (DO-178/DO-178A) software level, can it automatically considered 'equivalent to' a certain ED-12B / DO-178B level (and be installed on the product requiring that ED-12B / DO-178B level)?" For example, if the legacy system has ED-12A/DO-178A Level 2 software, it can be installed on a product requiring ED-12B / DO-178B Levels C, D, or E.

   There are two entries in Table 12-1 that may require additional analysis before determining equivalency; these instances are shown by an "Analyse" in Table 12-1. There should be an agreement between the certification authorities and applicant, when analysis is required.

If equivalency is not established by Table 12-1 (i.e., a "NO" entry in the table), the provisions of ED-12B / DO-178B Section 12.1.4 should be applied to the software application or partition to upgrade the software level. Procedures for applying Section 12.1.4 are not covered by this section of the Certification Memorandum. The remainder of this section of the Certification Memorandum assumes that equivalency has been established.

	Legacy System Software Level per ED-12/ED-12A		
ED-12B / DO-178B SW Level Required by the Installation	**Critical/Level 1**	**Essential/Level 2**	**Non-essential/ Level 3**
A	Possibly YES after Analysis	NO	NO
B	YES	NO/Analyse	NO
C	YES	YES	NO
D	YES	YES	NO
E	YES	YES	YES

Table 12-1 Software Level Equivalence

b. If the legacy system's software is unmodified and is being reinstalled on the same aircraft or engine or a different aircraft or engine where ED-12B / DO-178B is not required, then the original assurance process and associated data submittals may be accepted. This is only true if the system is being used in exactly the same way as originally installed in a certified product, has no added functionality since the original or subsequent certification approvals, and has not experienced service difficulties (e.g., Airworthiness Directives, Service Bulletins, etc.).

c. If the legacy system's software is modified and installed on the same aircraft or engine or on a different aircraft or engine where ED-12B / DO-178B is not adopted as the means of demonstrating compliance for software, then either the compliance means of the original aircraft or engine or the compliance means of the original legacy system may be used. Again, this is only true if the system is being used in exactly the same way as originally certified, has no added functionality since the original certification, and has not experienced in-service difficulties. A change impact analysis should be conducted to evaluate the software modifications and to apply appropriate regression testing.

d. If the legacy system software is modified and installed on different aircraft or engine where ED-12B / DO-178B is adopted as the means of demonstrating compliance, it should be assessed whether the change is a small, simple change (as discussed in paragraph 12.3d of this Certification Memorandum). Any changes determined to be small, simple changes may be handled in the same manner as the not modified case discussed in paragraph 12.4b of this Certification Memorandum. The determination of whether a change is a small, simple change shall be at the discretion of the certification authorities specialists involved or under the privileges of a Design Organisation Approval. Some representative, but not exhaustive examples, of small, simple changes are given in paragraph 12.3d of this Certification Memorandum.

(1) If the change is not a small, simple change, all the changes to the software and all of the components affected by the change should be assured using ED-12B / DO-178B (as discussed in paragraph 12.3f of this Certification Memorandum). The change impact analysis is the normal means of determining affected components. A description of change impact analysis is beyond the scope of this Certification Memorandum. However, the project plans and processes and the change activities and items of evidence should be shown to meet the objectives of ED-12B / DO-178B. For example, if the original software was not evaluated using the structural coverage criteria in ED-12B / DO-178B Section 6 and Annex A, then ED-12B / DO-178B verification specified for the software level of the changed software will have to be completed and the coverage criteria satisfied.

(2) Additional affected, but unchanged, components may not have to be evaluated for internal structural coverage but would have to meet the requirements for data coupling and control coupling coverage (such as to verify no changes to component interfaces with other components using integration testing), as well as requirements based test coverage for those affected functions. Once this process is complete, the applicant may be allowed to claim that their legacy system software application or partition complies with ED-12B / DO-178B at the certification authority's discretion, depending on the significance of the modifications and evidence produced.

e. If the legacy system software is not modified but is installed on a different aircraft or engine (i.e., with a different type certificate) where ED-12B / DO-178B is adopted as the means of demonstrating compliance, then there should not be a separate compliance finding for the software. The original certification serves as the installation certification of the software, unless the operational use of the system is expected to be significantly different (for example, an air data computer installed on piston-powered general aviation aircraft flying below 14,500 feet is now installed on a corporate jet flying at 50,000 feet). When the operational use is significantly different from the original certification basis, an assurance to ED-12B / DO-178B guidance should be performed. The determination of the significance in change of the operational use shall be at the discretion of the certification authorities specialists involved.

f. All changes to legacy systems and the process used to approve those changes should be documented in the Plan for Software Aspects of Certification (PSAC), Configuration Index Document (CID), and/or the Software Accomplishment Summary (SAS), as appropriate for the specific project. If service history is claimed for the legacy system, those data should be summarized in the SAS as well.

g. If any future changes are proposed, they should be addressed by using the criteria specified in this section of the Certification Memorandum.

13 OVERSIGHT OF SOFTWARE CHANGE IMPACT ANALYSES USED TO CLASSIFY SOFTWARE CHANGES AS MAJOR OR MINOR

13.1 BACKGROUND

ED-12B / DO-178B, Section 12.1.1, identifies analysis activities to be performed for proposed software changes. ED-12B / DO-178B also states that re-verification should be accomplished on all software changes and areas affected by those changes.

Subpart D of Part 21 addresses the classification of changes to type design as minor or major. Paragraph 21A.91 proposes criteria for the classification of changes to a type design as minor or major.

The purpose of this classification is to determine the certification route to be followed in Part 21 Subpart D (either 21A.95 or 21A.97) or alternatively in Subpart E.

13.2 PROCEDURES

Detailed guidance for the classification of changes to type design is given in GM 21A.91.

14 GUIDELINES FOR APPROVING REUSED SOFTWARE LIFE CYCLE DATA

14.1 PURPOSE

This section provides guidelines for determining whether software life cycle data, produced and approved for one certification project, can be approved on a follow-on certification project. Approval for reuse could minimize the amount of rework while maintaining an equivalent level of design assurance.

14.2 DISCUSSION

14.2.1 Software suitable for reuse

a. If properly planned and packaged, software life cycle data can be reused from one project to the next, with minimal rework. For example, the software plans, requirements, design, and other software life cycle data (as documented in a Software Configuration Index) for a System may originally be approved on System #1 (the original certification project) and reused on System #2 (the subsequent certification project). Sample items suitable for reuse include:

(1) **Software plans and standards.** These include software undergoing non-substantive changes, such as:

i. Program name,

ii. Name change due to consolidations or mergers, and

iii. Configuration changes for reasons other than design changes (for example, document format change, drawing modifications, or documentation system changes).

(2) **Tool qualification data.** The certification authorities can approve reuse, if the tool is used exactly as specified in the qualification approval as part of the original certification, and the applicant has access to the tool qualification data. This is true even if some of the features were qualified but not used during the original certification. The applicant should ensure that the same version of the tools is being used as that supported by the qualification data. The certification authorities will not approve reuse if the applicant uses additional or different tool functionality than was previously qualified.

(3) **Software libraries.** The certification authorities can approve library sets in the original certification project if the library set is used identically (that is, same library functions are used the same way).

(4) **Software requirements, design, code, verification procedures, and verification results.** The certification authorities may approve these for reuse after the applicant makes a thorough change impact analysis. This is to confirm that the requirements, design, code, procedures, and so forth are unaffected and unchanged from the previous certification effort.

(5) **Configuration items.** These may be approved for reuse in their entirety, if the certification authority and staff members responsible for certification use paragraph 14-3 of this section to make sure the determination, and the configuration of the software life cycle data has not changed. Configuration item requirements verified at a higher level (that is, system level) should be identified in the original configuration and re-verified before reuse.

b. Projects not using ED-12B / DO-178B may have additional considerations not documented in this section. Certification authorities should evaluate them on a case-by-case basis. The applicant should contact EASA for guidance.

14.2.2 Safety considerations

If the certification authorities find software life cycle data acceptable for reuse, no further design approval is required. Table 14-1 illustrates the considerations that govern whether the certification authorities will approve software reuse.

The certification authorities may approve for reuse if:	1. There is no adverse effect on original system safety margins, **and** 2. There is no adverse effect on original operational capability UNLESS accompanied by a justifiable increase in safety.
The certification authorities will NOT approve for reuse if the reuse:	1. Adversely affects safety, **or** 2. Exceeds a pre-approved range of data or parameters, **or** 3. Exceeds an equipment performance characteristic.

Table 14-1 Reuse Approval Considerations

14.2.3 Factors affecting reuse

a. Any of the software life cycle data items in Section 11 of ED-12B / DO-178B is suitable for reuse. To meet the guidelines in paragraph 14.3 of this section, the software life cycle data should be unchanged, and should apply to the project for which reuse is being considered.

b. In-service problems with previous applications can limit reuse. There may be Airworthiness Directives or a manufacturer's unresolved problem reports with the previously approved system. The applicant needs to analyze all open manufacturer's problem reports to ensure that the reusable portion of the new software is not affected. If the reusable portion of the new software is affected, changes to correct that software life cycle data should be made or the software should not be used.

c. Applicants should determine whether the software data apply to the subsequent project's development by assessing the similarity of both the operational environment and the software development environment. They should:

   (1) Assess the operational environment by evaluating the end-to-end performance requirements and the operational safety assessment.

   (2) Refer to the Software Life Cycle Environment Configuration Index in Section 11.15 of ED-12B / DO-178B, when assessing the software development environment.

   (3) Demonstrate that operational and development environments are the same, or demonstrated to produce identical results as the previous certification.

   (4) Assess any outstanding problem reports.

14.3 PROCEDURES

a. The certification authority should ensure that the applicant has met the following guidelines before granting certification credit for reused software life cycle data:

(1) The software life cycle data have not changed since its previous approval.

(2) The software level of the software application(s) is equal to (or less than) the software level of the original certification effort.

(3) The range and data type of inputs to the configuration item are equivalent to its approved predecessor.

(4) The configuration item is embedded on the same target computer and is used the same way operationally as the original certification project.

(5) Equivalent software/hardware integration testing and system testing were conducted on the same target computer and system as in the original certification project.

(6) The applicant followed the safety considerations and reuse factors in paragraphs 14.2.2 and 14.2.3 of this section.

(7) The software life cycle data and the rationale for reuse of each item are documented in the "Additional Considerations" portion of the PSAC. The applicant's PSAC should include method of use, integration, and documentation for the reused configuration item. The PSAC should be submitted as early as possible in the development program. The applicant should also document all references to the project previously certified and the project number, as applicable, in the PSAC.

The certification authority representative responsible for the subsequent certification should review the PSAC and notify the applicant whether the proposal is acceptable or not (with appropriate rationale).

15 PROPERLY OVERSEEING SUPPLIERS

15.1 BACKGROUND

a. Many TC/STC/ETSOA applicants have shifted system and software development, verification, and certification activities onto their aircraft system suppliers and sub-tier suppliers. In the past, these suppliers participated in compliance activities only at their respective system, subsystem, or component levels. With airborne systems becoming increasingly more complex and integrated, and suppliers and sub-tier suppliers accepting these new responsibilities, EASA is concerned that their lack of expertise could result in incomplete or deficient certification activities.

b. Each responsibility that the applicant delegates to a supplier creates an interface with that supplier that needs to be validated and verified to ensure that the transition from the supplier's processes to the applicant's processes (or vice-versa) is accomplished correctly and accurately. Lack of proper validation and verification of life cycle data at the transition point has resulted in issues with regard to requirements, problem reporting, changes, etc.

c. Finally, retention of substantiating data, such as software life cycle data and other certification and compliance data, is a critical part of the certification process. When this data is retained by a sub-tier supplier, it may not be readily available to us. This may also affect the continued operational safety of the aircraft and its systems, especially with regard to in-service problems (service difficulties), problem resolution (service bulletins), and mandatory corrections (airworthiness directives).

15.2 EASA CERTIFICATION POLICY

15.2.1 Supplier oversight aspects in plans and procedures

The applicant should create oversight plans and procedures that will ensure all suppliers and sub-tier suppliers will comply with all regulations, policy, guidance, agreements, and standards that apply to the certification program. The applicable publications include, but are not limited to:

(1) EASA Certification Specifications;
(2) EASA CRIs and Certification Memoranda;
(3) Applicant DOA procedures, airworthiness representative procedures, and memoranda of agreement;
(4) Applicant standards for software development (including requirements, design, and coding standards);
(5) Applicant quality assurance plans, procedures, and processes;
(6) Applicant configuration management plans, procedures, and processes;
(7) System supplier standards, plans, procedures and processes.

The applicant's planning documents, such as certification plans and Plans for Software Aspects of Certification (PSACs), should describe how the applicant will have visibility into their suppliers' and sub-tier suppliers' activities. This includes commercial off-the-shelf software component suppliers and vendors. The applicant should submit these plans for review and approval, preferably early in the program. The applicant should avoid making changes to the plans late in the program. If late changes are unavoidable, the applicant must allow adequate time for review and consideration.

15.2.2 Supplier oversight: reviewing the applicant's plans

The applicant should address the following concerns in a supplier management plan or other suitable planning documents. Certification Specialists should review the plan(s) and ensure that the following areas are addressed to their satisfaction:

1. Visibility into compliance with regulations, policy, plans, standards, and agreements: The plan should address how the applicant will ensure that all applicable regulations, policy, plans, standards, CRIs, and memoranda of agreement are conveyed to, coordinated with, and complied with by prime and sub-tier suppliers.

2. Integration management: The plan should address how the system components will be integrated, and who will be responsible for validating and verifying the software and the integrated system. The plan should address:

    (a) How requirements will be implemented, managed, and validated; including safety requirements, derived requirements, and changes to requirements;

    (b) How the design will be controlled and approved;

    (c) How the integration test environment will be controlled;

    (d) How the software build and release process will be controlled (reconcile any differences between the supplier's and the applicant's release strategies);

    (e) What product assurance activities that support the certification requirements will be conducted and who will be conducting them; and

    (f) The applicant's strategy for integrating and verifying the system, including requirements-based testing and structural coverage analysis.

3. Tasks and responsibilities: The plan should identify who the responsible persons are and what their responsibilities are, who the focal points are, and how their activities will be coordinated and communicated. It should identify who will approve or recommend approval of software life cycle data.

4. Problem reporting and resolution: The plan should establish a system to track problem reports. It should describe how problems will be reported between the applicant and all levels of suppliers. The problem reporting system should ensure that problems are resolved, and that reports and the resulting changes are recorded in a configuration management system. The plan should describe how the responsible person(s) will oversee problem reporting.

5. Integration verification activity: The plan should identify who will be responsible for ensuring that all integration verification activities between all levels of suppliers comply with applicable guidance. It should describe how the responsible person(s) will oversee the verification process.

6. Configuration management: The plan should describe the procedures and tools to aid configuration management of all software life cycle data. It should describe how configuration control will be maintained across all sub-tier suppliers, including those in foreign locations, and how responsible persons will oversee configuration management.

7. Compliance substantiation and data retention: The plan should describe how the applicant will ensure that all supplier and sub-tier supplier compliance findings are substantiated and retained for the program. The plan should address, at minimum, the following certification data:
    (a) Evidence that compliance has been demonstrated;
    (b) Verification and validation data; and

(c) Software life cycle data.

The applicant's supplier management plan (or equivalent plans) should address the concern identified in paragraph 15.1.b. regarding the transition of life cycle data between the applicant's processes and the suppliers' processes. The plan should address the validation and verification of data with regard to all processes, including requirements management, problem reporting, use of standards, change impact, reviews, etc.

16 MANAGEMENT OF PROBLEM REPORTS

16.1 BACKGROUND

Problems related to airborne software may surface relatively late in the industrial development process. When these problems do not affect the safety of the aircraft/engine (and compliance with the objectives of the EASA Certification Specifications has been demonstrated), the applicant may decide to propose for certification some items of airborne software that still have known problems.

For airborne software, this situation is covered by ED-12B/DO-178B, section 11.20(j), as follows:

"Software status: this section [of the Software Accomplishment Summary - SAS - document produced for certification] contains a summary of problem reports unresolved at the time of certification, including a statement of functional limitations."

Problems may arise due to the methods that are used by the suppliers and sub-tier suppliers of applicants for tracking and reporting problem reports. There may be inconsistencies in the reporting and tracking of problem reports and the tools that are used to track them between the applicant, their suppliers and their sub-tier suppliers. This may make it difficult for the applicant and the certification authority to gain an accurate picture of the number and the severity of the open problem reports across the various groups that are involved.

The use of suppliers and sub-tier suppliers may also result in situations where the sub-tier suppliers do not have sufficient knowledge and visibility of system level requirements and considerations when evaluating problem reports and their effects. The intent of this section of this software Certification Memorandum is to discuss the issues related to Problem Management for airborne software.

16.2 OBJECTIVES

One of the principal objectives of any airborne software development and certification should be to minimise the number and the severity of Open Problem Reports (OPRs) in any airborne software release that is proposed for certification. The OPR management principles and assessment guidelines detailed in this section of this Certification Memorandum should not, in any case, be understood as a justification for an applicant to deviate from this prevailing objective.

This section of this Certification Memorandum has three purposes:

1. To clarify the role of the aircraft/engine manufacturer and the equipment supplier in the assessment of limitations of a piece of equipment with embedded airborne software because of known problems at the time of certification. It should be noted that even if the equipment manufacturer has sufficient knowledge to explain the functional effect of an OPR on the equipment/item, only the aircraft/engine manufacturer can assess or confirm the potential effect at the system/aircraft/engine level.
2. To facilitate the assessment of the acceptability of a baseline released with Open Problems reports, by defining a harmonized categorization of OPRs and an adequate means of recording the category of an OPR.
3. To clarify the aspects of problem reporting that should be covered in the plans of suppliers and sub-tier suppliers of the applicant.

16.3 SCOPE

This section of this Certification Memorandum is applicable to all systems containing digital equipment with a Development Assurance Level (DAL) of A, B, C or D.

16.4 TERMINOLOGY

The text in italics in the definitions below is extracted from the glossary of ED-12B/DO-178B.

- **Problem**: any of the following features: error, fault, failure, deviation from the rules
- **Error**: *a mistake in requirements, design or code.*
- **Fault**: *(1) A manifestation of an error in software. A fault, if it occurs, may cause a failure.*
- **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.* But a fault may also remain hidden at system level and have no operational consequences.
- **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 […].*
- **Deviation from the rules**: a non-conformity of the development process with the plans, development standards, applicable CRIs. In the particular case where a non conformity with the plans or development standards is intentional and the plans or development standards are planned to be modified accordingly, such a non-conformity might not be recorded as a problem, but instead be identified and justified in the compliance status of the HAS.
- **Open Problem Report (OPR)**: a problem which has not been corrected at the time the airborne electronic hardware is presented for certification.

16.5 TYPOLOGY OF OPEN PROBLEM REPORTS

A logged OPR should be categorized according to the nature and effect of the OPR. One possible way to classify OPRs that is acceptable to EASA is as follows:

- **Type 0**: a problem whose consequence is a failure – under certain conditions - of the system, with a safety impact.
- **Type 1**: a problem whose consequence is a failure – under certain conditions - of the system, having no safety impact on the aircraft/engine. (This needs to be confirmed by the aircraft/engine manufacturer). If agreed between the aircraft/engine manufacturer and the equipment/software supplier, this type should be divided into two sub-types:
  - **Type 1A**: a failure with a "significant" functional consequence; the meaning of "significant" should be defined in the context of the related system in agreement between the aircraft/engine manufacturer and the equipment/software supplier (for instance a "cockpit effect").
  - **Type 1B**: a failure with no "significant" functional consequences.
- **Type 2**: a fault which does not result in a failure (i.e.: no system functional consequences and the fault is not detectable by the crew in any foreseeable operating conditions).
- **Type 3**: Any problem which is not of type 0, 1 or 2, but which is a deviation from the rules (i.e. the plans or software development standards, applicable CRIs). If agreed between the aircraft/engine manufacturer and the equipment/software supplier, this type should be divided into two sub-types:

- **Type 3A:** a "significant" deviation, whose effects could be to lower the assurance that the airborne software behaves as intended and has no unintended behaviour.
- **Type 3B:** a "non-significant" deviation from the methodology (plans) that does not affect the assurance obtained.

16.6 GUIDELINES ON OPR MANAGEMENT

EASA considers that, as far as possible, a root cause analysis should be performed for all OPRs, except in exceptional cases where a root cause analysis is not feasible. Any such infeasibility should be justified. In the cases of Types 0, 1 or 2, this root cause analysis should lead to the identification of the corresponding error in the code and of any associated methodological deviations. For type 3 problems, the root cause analysis consists of the identification of the methodological deviation associated with the problem.

All OPRs should be categorized according to the typology of problems defined in this CRI, or an equivalent typology. If an equivalent typology is proposed, any new type(s) should correspond to only one of the types (0, 1, 2 or 3) as defined in this section of this Certification Memorandum.

All OPRs should be described in order to substantiate their categorization into adequate types; this description should be recorded.

When previously developed airborne software is used, previously existing OPRs should be reassessed in the operational environment of the aircraft/engine to be certified.

In order to avoid decreasing the assurance of the quality of the airborne software to be certified due to an increasing number of OPRs, the following objectives should be taken into account and acted upon:

- Limitations should be removed at the earliest opportunity.
- Conformity with the specifications should be restored at the earliest opportunity.
- Any OPR should be rectified within a time period compatible with its assessed consequences.

Per ED-79/ARP4754 section 9.2.2, problem reporting should be managed at the system level.

The following type-based objectives should be taken into account:

- Type 0: such OPRs should be rectified before certification or an adequate means of mitigation (for instance, operating limitations,) should be proposed such that there are no adverse effects on safety at the aircraft/engine level.
- Type 0 and 1: Potential effects should be assessed at the system level and, if necessary, at the aircraft/engine level. If necessary, appropriate limitations should be defined in order to ensure there are no adverse effects on safety.
- Type 1: Any claim that an OPR has no safety impact on the aircraft/engine should be justified; this justification should be recorded.
- Type 2: The justification that the error cannot cause a failure should be recorded. For simple cases, this justification may be a simple statement based on engineering judgement. In some specific cases, this justification may imply that some specific additional validation and/or verification activities need to be performed.

16.7 CONTENTS OF SOFTWARE ACCOMPLISHMENT SUMMARY (SAS)

All OPRs of types 0 to 3 should be recorded in the SAS or the equivalent certification document, along with the following information:

- Supplier's identification of the OPR (configuration management number)
- Type of OPR

- Is the function of the system affected? (Provide a description if "Yes", or the Statement "No")
- Short description (including a brief summary of the root cause, where available)
- Date when the OPR was opened
- Scheduled closure date for the OPR
- Brief justification as to why it can be left open
- Means of mitigation to ensure there are no adverse safety effects - if applicable
- Interrelationships between OPRs - if applicable.

Although a limited number of type 2 or 3 OPRs should normally not prevent certification, a large number of type 2 or 3 OPRs, or a lack of action plans for the closure of type 2 and 3 OPRs are general indicators of a lack of software assurance. The EASA team may reject a request for certification if the number of remaining OPRs is too high, or if there is no evidence of an adequate action plan to close the OPRs.

16.8 CONTENT OF SYSTEM CERTIFICATION SUMMARY OR EQUIVALENT DOCUMENT

The System Certification Summary or an equivalent certification document should describe:

- The identification of all type 0 and 1 OPRs and the description of their impact at the system level or, if necessary, at the aircraft/engine level (including, any associated operational limitations and procedures).

16.9 OVERSIGHT OF PROBLEM REPORTING

16.9.1 Problem reporting and supplier plans

In order to ensure that software problems are consistently reported and resolved, and that software development assurance is accomplished before certification, the applicant should discuss in their Software Configuration Management Plan, or other appropriate planning documents, how they will oversee their supplier's and sub-tier supplier's software problem reporting process. The engineer responsible for certification should review the plans and verify that they address the following to their satisfaction:

1) The plans should describe each of the applicant's supplier's and sub-tier supplier's problem reporting processes that will ensure problems are reported, assessed, resolved, implemented, re-verified (regression testing and analysis), closed, and controlled. The plans should consider all problems related to software, databases, data items, and electronic files used in any systems and equipment installed on the aircraft.

2) The plans should establish how problem reports will be categorized so that each problem report can be classified accordingly. The categories described above should be used.

3) The plans should describe how the applicant's suppliers and sub-tier suppliers will notify the applicant of any problems that could impact safety, performance, functional or operational characteristics, software assurance, or compliance.
   a) The applicant may enter such problems into their own problem reporting and tracking system. If so, the plan needs to describe how this is accomplished. If the supplier's problem reporting system is not directly compatible with the applicant's system, the plan needs to describe a process for verifying the translation between problem reporting systems.
   b) The applicant may allow their suppliers and sub-tier suppliers to have access to their own problem reporting system. Doing so may help the applicant ensure that they will

properly receive and control their supplier's problem reports. If the applicant allows this, they should restrict who has such access in order to maintain proper configuration control, and their suppliers should be trained on the proper use of the reporting system.

c) The plans should describe any tools that the applicant's suppliers or sub-tier suppliers plan to use for the purpose of recording action items or observations for the applicant to review and approve prior to entering them into the applicant's problem reporting system.
d) The plans should state that suppliers will have only one problem reporting system in order to assure that the applicant will have visibility into all problems and that no problems are hidden from the applicant.
e) Any problems that may influence other applications, or that may have system-wide influence should be made visible to the appropriate disciplines.

4) The plans should describe how flight test, human factors, systems, software, and other engineers of the appropriate disciplines will be involved in reviewing each supplier's and sub-tier supplier's problem report resolution process. They should also describe how these engineers will participate in problem report review boards and change control boards.

5) The plans should establish the criteria that problem report review boards and change control boards will use in determining the acceptability of any open problem reports that the applicant will propose to defer beyond certification.

   a) These boards should carefully consider the potential impacts of any open problem reports on safety, functionality, and operation.
   b) Since a significant number of unresolved problem reports indicate that the software may not be fully mature and its assurance questionable, the applicant should describe a process for establishing an upper boundary or target limit on the number of problem reports allowed to be deferred until after type certification.
   c) The plan should establish a means of determining a time limit by which unresolved problem reports deferred beyond certification will be resolved. This applies to problem reports generated by the applicant, suppliers, and sub-tier suppliers.

16.9.2 Reviewing open problem reports

The person responsible for certification should be involved in certain decisions related to open problem reports prior to certification and should:

1) Review, as appropriate, any problem reports that are proposed for deferral beyond certification. This review may require EASA flight test, systems, and other specialists and may require more information to make the assessment. Where there are concerns that safety might be impacted, the deferral of specific problem reports may be disallowed.

2) If the applicant is using previously developed software, ensure that the applicant has reassessed any open problem reports for their potential impact on the aircraft or system baseline to be certified.

3) Ensure that the applicant has considered the inter-relationships of multiple open problem reports and assessed whether any open problem report has become more critical when considered in conjunction with another related problem report.

4) Ensure that the applicant has reviewed any open problem reports related to airworthiness directives, service bulletins, or operating limitations and other mandatory corrections or conditions. The applicant may need help to determine which problems to resolve before certification.

5) Review any open problem reports with potential safety or operational impact to determine if operational limitations and procedures are required before EASA test

pilots participate in test flights. Other technical experts should be involved as necessary in making this determination.

6) Ensure that the applicant has complied with ED-12B / DO-178B, section 11.20(j).

17 EMBEDDED SOFTWARE CONFIGURATION FILES

17.1 BACKGROUND

Databases may be used in various aircraft/engine systems and be embedded in pieces of digital equipment:

- Some databases may be used to store aeronautical data such as navigation or terrain data, magnetic variation data, or data produced by the applicant, such as aircraft performance parameters for use by aircraft systems such as navigation systems, surveillance systems, etc. Such Databases are usually defined as **Aeronautical Databases** are not covered in this Certification Memorandum.
- Other databases are segregated from the rest of the embedded software for modularity and portability purposes, such as data for symbology, bus specifications or A/C configuration databases. These Databases, referred as configuration files, may be used to activate or deactivate software components/functions, to adapt the software computations to the aircraft configuration, or may be used as computational data. With such Databases, embedded software may be adapted to operate in any one of several aircraft/engine configurations. They may, for example, embed Boolean flags for software "Pin Programming" and / or data parameters for the adaptation of the operational software to the aircraft installation and any installation options. Such Databases are usually defined as **Configuration Files (CF).** These configuration files are the subject of this section of the software Certification Memorandum.

Most databases or Configuration Files may be loaded on the aircraft and may be updated after delivery without necessitating a change to the related digital equipment. Such databases and Configuration Files are usually controlled by their own part number or by aircraft modification numbers, independently of the part numbering of the associated digital equipment. In addition, they usually have the capability of being loaded without removing the system or equipment from its installation (in which case the file is 'field loadable'). Data loaders or portable devices are used to store the Databases or Configuration Files in the computer using Non Volatile Memory so that they may later be used by the operational software.

Any corruption or error in the format or content of databases, or any incompatibility between databases and aircraft systems, may have an impact on safety and should be considered when showing compliance with the safety objectives[2] of the Certification Specifications (CS) for those activities for which the applicant is responsible. The intended function of a system can be affected by the quality of the data contained in the database.

This section of this Certification Memorandum applies to pieces of digital equipment whose software is composed of a set of executable files (containing operational software) and a set of configuration files with the following characteristics: the files are -

- Identified by a specific P/N,
- Designed to be updated without necessarily requiring a change to the associated executable software.

The operational software should also be designed to allow an update of the Configuration Files.

This section of this Certification Memorandum does not apply to configuration files with a P/N that is the same as that of the executable software. Such configuration files are certified as part of the executable software by using ED-12B/DO-178B, which adequately addresses the development (including verification) of such embedded configuration files and the associated executable software.

[2] CS 25.1309 for Large Aeroplanes, CS 23.1309 for Small Aeroplanes, CS27.1309 for Small Rotorcraft, CS29.1309 for Large Rotorcraft and CS E-50 (f) for engines.

On the other hand, ED-12B/DO-178B does not clearly address the development of configuration files when they are managed separately from the associated executable software. This section of this Certification Memorandum aims to establish acceptable guidance material for the development, validation, verification and utilisation of such configuration files on board an aircraft/engine. Some requirements of this section of this Certification Memorandum also address additional guidance for the associated executable software.

Such configuration files will, in most cases, be field-loadable (see the definition below). Certification issues and the required safety precautions associated with field loading are further discussed in the specific section on Field Loadable Software of this Certification Memorandum.

17.2 IDENTIFICATION OF CONFIGURATION FILES

The applicant is requested to provide a list and a description of all the configuration files which will be used in the pieces of digital equipment of the aircraft/engine. These lists may be issued system by system and included in the certification plans or at the software level in the Plan for Software Aspects of Certification (PSAC).

In this list, the applicant should identify the configuration files that will have a separate part number and separate approval of the files in order to allow them to be updated without necessarily requiring a change to the associated executable software.

The intent of the requirements stated hereafter is to identify the ED-12B/DO-178B objectives that are applicable to these Configuration files. Compliance with the ED-12B/DO-178B requirements referred to hereafter should be demonstrated taking into account the development assurance level (DAL) allocated to the configuration files. The applicability of these requirements is dependent on this DAL level in the manner described in the tables of objectives in ED-12B/DO-178B. (For example, some objectives of ED-12B/DO-178B Table A-3 are not required for DALs C and D). The documentation to be provided should be according to the guidelines defined below.

In addition, specific requirements (in addition to those specified in ED-12B/DO-178B) are also identified hereafter for the operational software that is associated with these Configuration Files.

17.3 DEVELOPMENT ASSURANCE LEVEL

Configuration files should be integrated into the system development as a particular item in a similar manner to that used for a software component (see ED-79/ARP4754). The system safety assessment should include the determination of the failure condition criticality associated with incorrect data or loss of data of each Configuration File. In other words, the risk of an unintended and undetected activation of a function due to a data error should be assessed as part of the system safety assessment process. The CF should then be allocated a Development Assurance Level between A and E, depending on the outcome of the safety analysis of the system, as indicated in ED-79/ARP4754.

The requirements stated hereafter are applicable only for Configuration Files of DALs A to D that are updated on the ground by maintenance operators or crew.

17.4 IDENTIFICATION AND CONFIGURATION CONTROL

- The CF should be identified (Specific P/N).
- The configuration of the CF should be managed.
- For any CF that has an electronic P/N, the electronic identification of the CF may have to be included as part of the configuration data. (See also the specific section on Field Loadable Software in this Certification Memorandum.)

17.5 DATA QUALITY

The development of configuration files should follow the requirements stated hereafter. These requirements originate from ED-12B/DO-178B requirements, which are cross-referenced whenever possible. Some requirements may impact the design and development activities for the executable software.

- **Planning process:** The applicant should document the CF development process that is used so as to take into account the requirements stated hereafter. The documentation may be specific or may be part of the executable SW documentation**.**
- **Description**: A document should describe the structure of each configuration file and the usage of its data, including the types and ranges of the data. This document will be referred to hereafter as the 'CF design specification'. This document will be equivalent to the Software Requirements Specification data of the executable software. This document should also be managed as an item of CC1 data (see ED-12B/DO-178B section 7).
- **Validation**: The CF design specification should be validated (to be accurate, consistent, verifiable, correct, complete – as described in ED-12B/DO-178B section 5.1). All items from ED-12B/DO-178B table A-3 (covering the verification of the outputs of the requirement process) should be assessed for applicability.
- **Verification**: The implementation of the CF should be verified against the CF design specification (see also ED-12B/DO-178B Table A-6). All items from ED-12B/DO-178B table A-6 (testing of outputs of integration process) should be assessed for applicability.
- **Deactivated code**: Some values in the CF may result in the deactivation of certain sections of code in the executable software. ED-12B/DO-178B requirements (mainly sections 4.2.h, 5.4.3.a, 5.4.3.b, 6.4.4.3.d of ED-12B/DO-178B) concerning any such deactivated code should be considered for the executable software. The activation or deactivation of a function, through the parameter values in the configuration files, should not affect the behaviour of any other function (see also ED-12B/DO-178B section 2.4.e).
- **CF modification without modification of the executable software**: In addition to CF activities, assurance should be given (through documented justifications such as analysis and/or tests) that the performance of the executable software (e.g. timing requirements, memory and stack margins, communication and scheduling) is not affected beyond specified limits (as defined in the development process of the executable software). In cases where these performance limits are exceeded, the development and verification life cycles of the executable software should be re-entered in order to perform all the activities that are necessary: for example, the executable software might have to be modified and re-tested so as to be compatible with the new CF.
- **Documentation:** for each identified CF, the means to fulfil the following objectives should be documented:
  - configuration management
  - description of the development process
  - accomplishment summary
  - configuration index document

  The certification documents affected by a modification of a Configuration File should be identified in the system certification plan, which should be reissued as necessary.

17.6 COMPATIBILITY / MIXABILITY

Each Configuration File (CF) should be considered as a separate configuration item with the same compatibility/mixability concerns as any software configuration item.

The following requirements are applicable to the executable software -

- Assurance should be provided that the executable software does not use any corrupted or non-acceptable CF (where "acceptable" means that the considered Configuration File is compliant with its specified format and parameter ranges). For this purpose, appropriate means should be proposed that are able to ensure the required level of safety, such as automatic detection mechanisms and/or maintenance procedures.
- Assurance should be provided that the associated executable software is developed, validated and verified for any CF to be certified.
- Compatibility of the load: When the CF requires a specific executable software version, the compatibility between the executable software and the Configuration File has to be ensured.

  Note: A means which could detect any inappropriate combinations of software and/or hardware and/or airframe and revert to a certified configuration in the event of detecting any incorrect combinations would cover partially this concern. For other topics linked with compatibility or integrity, refer to the specific section on Field Loadable Software in this Certification Memorandum.

The following requirement is applicable to Configuration Files (CF).

- CF evolution: the new configuration (executable SW + CF) should be identified in the CF Accomplishment Summary, which should include a summary of the validation and verification activities performed. The accepted associations / compatibilities between the CF and the executable software should be described in documents produced at the Configuration File level or at the system level.

In cases where CFs are used to activate several functions or components ("software pin programming"), the applicant should identify the procedures they propose to put in place to ensure that the functions or components to be activated have been properly certified prior to the release of the CF, and that the compatibility/mixability issues associated with it have been properly controlled. The certification documents affected by such CFs should be identified and reissued as needed.

17.7 GENERATION OF CONFIGURATION FILES

In cases where a tool is used to produce the CF, the need for tool qualification should be considered, taking into account the recommendations of ED-12B/DO-178B (section 12.2).

18 MANAGING THE SOFTWARE DEVELOPMENT AND VERIFICATION ENVIRONMENT

18.1 BACKGROUND

ED-12B / DO-178B requires that the verification test activities take place on the target computer, a target emulator, or a host computer simulator. Software development and verification teams typically utilize an environment designed specifically to emulate the target computer to satisfy this requirement. Because the environment may go through several iterations during software development and verification, it may not be clear how representative the environment is of the actual production hardware at any point in time in the verification process.

Additionally, the environment may not be identical to the final production version of the hardware to be installed in the aircraft. Therefore, the applicant should establish and maintain configuration control of the environment, and implement a structured problem reporting system for the environment available to users of the environment.

18.2 CONTROLLING THE DEVELOPMENT AND VERIFICATION ENVIRONMENT

The applicant should address the following aspects in their Software Development Plan, Software Verification Plan and Software Configuration Management Plan.

The applicant should convey these aspects to all participating software suppliers, and ensure that they comply with them. The person responsible for certification should review these plans and assess their adequacy.

The Software Verification Plan should include:

1) A description of the software development and verification environment, and an explanation of the differences between it and the production version of the system hardware and software to be installed on the aircraft.

2) An analysis for each identified difference (cf. (1)) and a justification that for each objective (cf. (3)), that the difference does not prevent the ED-12B / DO-178B objective from being fulfilled.

3) An explanation of how the software development and verification environment will be used by system software suppliers and with which ED-12B / DO-178B objectives it will be used to show compliance.

4) An explanation of how the software development and verification environment will be used to show compliance with ED-12B / DO-178B objectives that involve verification of the software executable object code. This should address the entire executable object code, not just individual functional software components. If development tools are being used in the integrated environment, then verification should also be performed in the integrated environment.

5) A process for analyzing completed verification activities and assessing the need to repeat any of those activities after changes are made to the software development and verification environment. The process should ensure that all affected verification activities will be repeated, or ensure that a documented analysis is conducted showing why retesting is not required.

The Software Configuration Management Plan should include:

1) A description of the configuration control system to be used for the software development and verification environment. The plan should identify the person who is responsible for administering this system.

2) A problem reporting and assessing system for the software development and verification environment that is available to all users of the environment (see section 16 of this Certification Memorandum).

19 THE USE OF OBJECT ORIENTED TECHNIQUES AT THE DESIGN OR SOURCE CODE LEVEL

19.1 BACKGROUND

Object-oriented techniques (OOTs) have several features that can make them undesirable for use in airborne systems if specific precautions are not taken and/or specific limitations are not imposed. For systems whose embedded software is to be designed using OOTs at the design level or the source code level, it might not be obvious how compliance with ED-12B / DO-178B can be achieved.

The use of OOTs also results in some new areas of concern that are specified in this section of this Certification Memorandum.

19.2 GUIDANCE

The OOT features listed below, if not properly controlled and verified, can result in software code that is non-deterministic, unused, and difficult to verify and maintain and whose configuration can change depending on the run-time state of the system.

The applicant should address the 17 background concerns (referred as Items) of this section of this Certification Memorandum.

The applicant should provide substantiating software plans, standards (requirements, design and code) and life cycle data to show that each of these concerns will be controlled for each system intended to use these OOTs on the aircraft/engine project and to ensure that an appropriate level of assurance is achieved for each.

The following areas of concern have been identified as being relevant to OOTs at either the design or source code level:

(a) - Software life cycle

As an adaptation to the growing complexity of systems and to the growing number of components and functions on the last generation of airborne software, OOTs could provide a new vision of the classical waterfall development process. By isolating functions in classes (even if they are involved in different requirements), it is possible to build and upgrade successive prototypes of the product. This process is known as 'Iterative Prototyping' or sometimes 'Fast Iterative Prototyping' and is beginning to be used more frequently. Without the application of a minimum set of rules, there is a risk of getting lost in a maze of upgrades, modifications and corrections, with the consequence of losing the capability to trace a code sequence to its requirement.

As applicable, the following items should be addressed:

- **Item 1:** The kind of iterative process (classical, iterative, etc).
- **Item 2:** Definition of the exit control process for each iteration (review, checklist, analysis).
- **Item 3:** Definition, hierarchy, granularity and coupling of classes, subclasses, method, attributes, etc.
- **Item 4:** Development and verification tool capabilities to reach the ED-12B/DO-178B objectives with regards to OOTs (functional areas, usage of features, capability, etc).
- **Item 5:** Specific complex features used in object-oriented languages compilers/ linkers/ builders and associated libraries (mathematics, semantics, etc.).

(b) - Traceability from function oriented requirements to object-oriented design or code

Traceability is the evidence of a link between a requirement, its implementation and test cases, etc.

It is used to identify derived requirements and architectural design decisions made during the software design process and to assess the implementation. Between the low-level requirements and the source-code, it allows verification of the absence of undocumented source code and that all low level requirements are verified.

Both classical or OOTs mentioned should address this concept, but object-oriented methods add a new area of concern. A system is designed from its objects instead of its functions. More precisely, objects are considered as the actors and the functions as some messages or events between the actors. But, in general, a system is described (in natural language) from a functional point of view. When a requirement engineer captures the functional behaviour of a system in an object-oriented view, it is likely that most of the required traceability will be lost.

As a possible means to overcome this problem, but not the only means, the SW team could use the following OOT features: use-cases in conjunction with dedicated diagrams.

- **Use-cases are built at the first step of the requirement analysis** and are typically used to specify and characterise the functionality of the software interacting with one or more external actors. Use-cases are developed to capture the software requirements and to understand what they are supposed to do and how their components interact together and with the outside.
- **Additional diagrams** are often used in conjunction with use cases: interaction diagrams, sequence diagrams, collaboration diagrams, state-chart diagrams, activity diagrams, class diagrams, component diagrams, deployment diagrams. These diagrams allow traceability, maintainability, portability, etc. The use of these diagrams depends on SW (complexity, real time constraints, interaction with external inputs or SW, number of class or use cases, OOTs, state machines, etc.). Usually, the complexity of a diagram is directly linked to the precision, focus and conciseness of a requirement.

As an example, criteria for a good OOT use-case could be:

- Completion of the events,
- Use driven,
- Simple enough to understand.

Use-cases with diagrams allow traceability from a functional description of software to an object-oriented design and therefore naturally promote traceability from the requirements down to the code.

The polymorphism capability of some OOT languages is also an issue for traceability that has to be addressed. Actually, if the same function name may lead to several different implementations depending on the context, this will increase the traceability workload in order to avoid wrong function call situations.

As applicable, the following items should be addressed:

- **Item 6:** Traceability between functional requirements and object design and/or object source code.
- **Item 7:** Definition of use case granularity.
- **Item 8:** Description of the usage of diagrams.

- **Item 9:** Specific additional analysis to complete the source to object code traceability for Level A SW.

(c) - Test coverage assessment (software requirements coverage assessment and structural coverage analysis)

Some concerns remain about the fulfilment of test coverage objectives described in ED-12B/DO-178B section 6.4:

> With the hierarchy capability of certain OOTs, a class may inherit from one or several other classes. This means that an operation of an ancestor is propagated to its descendants. Dynamic binding will convert at the very last moment which operations will have to be executed, depending on the instantiated class. In this environment, the result may be a polymorphic call. The functional coverage and the real time constraints of a class may then not be ensured.

The following concerns remain:

- The evaluation of the <u>functional</u> coverage (including the propagated methods).
- The structural coverage analysis.
- The flow analysis.
- The timing analysis.
- At a low level, to exercise requirements based tests for each class and method (including for instantiated derived classes).

Furthermore, with the possible use of the OOT encapsulation capability, sequences of code addressing a specific class may only have access to the public methods and data exported by the class. Programmers that implement a class have no access to the internal data of the class and may exercise unintended functionality if the description of the class is incompletely documented. The programmer may have no sight of what is really executed at lower levels of the class. Possible means (but not the only means) to prevent that could be:

- The class definition should be well documented in term of side effects, property bounds, pre-conditions and post conditions.
- Each method of a class (included hidden methods) should be considered as a software unit with its requirement and test cases.
- Functional, performance and robustness test cases should exercise all the encapsulated items, operations and attributes with structural coverage criteria depending on the software level.
- Regression activities (tests, analysis, etc.) should take into account the above items.

The use of the above-described diagrams may be helpful to support the definition of test cases and procedures.

As applicable, the following items should be addressed:

- **Item 10**: Strategy to achieve functional coverage (including the timing and flow analysis).
- **Item 11:** Strategy to achieve structural coverage (including the flow analysis).
- **Item 12:** Strategy to ensure the robustness behaviour of the SW.
- **Item 13:** Worst case demonstration for CPU workload and memory usage.
- **Item 14:** Detailed description of regression analysis (particularly for post TC).

(d) - Configuration management (Reuse of numerous classes)

Classes are generally designed to match definite objects (data buses, messages, screens, keyboard, etc.). They are generally built in order to ensure easy reuse of the classes in future applications. Some classes are purely abstract and are only bases for further specialised classes. In this context, the applicant should consider that:

- Parts of the classes will be considered as deactivated code. Software engineers will have to assess and give evidence that this deactivated code is not dead code. In any case, the software planning process should describe how the deactivated code (for classes, methods, attributes, etc. that are not used) would be defined, verified and handled to achieve the system safety objectives.

As applicable, the following items should be addressed:

- **Item 15:** Compliance with ED-12B/DO-178B objectives regarding deactivated code.

(e) – Other areas

OOT has several features that, if not controlled properly, may lead to unknown configurations of object code and to unverified code.

Moreover, many OOT features, if not properly controlled and verified, can result in software that is unpredictable, unused, and difficult to verify and whose configuration can change depending on the run-time state of the system:

Dynamic Binding/Dispatch: The matching of calls to methods (functions) at run-time as opposed to compile-time or link-time. This results from a polymorphic call. There are a number of concerns regarding the use of dynamic binding/dispatch in airborne software.

a) It complicates the flow analysis (e.g., data coupling and control coupling) and structural coverage analysis;
b) It can lead to complex and error-prone code;
c) It can complicate source code to object code traceability;
d) The matching of calls to methods can be difficult, if implicit type conversion is used; and
e) The behaviour of the execution of the compiler-generated code may not correspond to the expectations of the programmer.

Inheritance: A mechanism whereby a class is defined in terms of others (its parents), adding the features of its parents to its own. A class may have a single parent (which is known as 'single inheritance') or multiple parents (known as 'multiple inheritance'). Either the interface, or the interface and the implementation can be inherited. Where multiple inheritance is allowed, repeated inheritance is a possibility (two or more parents have a common ancestor in the class hierarchy). Multiple inheritance is particularly a concern in airborne systems. It can lead to overly complex and potentially unpredictable interactions between classes. It can also complicate traceability and verification.

Polymorphism: The ability of a name in software text to denote, at run-time, one or more possible entities, such as a function, a variable or an operator. For example, given the text: f(x), determining which f() should be called may be dependent on which class x belongs to, and x may belong to multiple classes, depending on the run-time state of the system. Polymorphism is generally supported by dynamic binding/dispatch. The concern with polymorphism in airborne systems is the potential for ambiguity, which might lead to coding errors, traceability problems, complexity, and difficulties for verification.

As applicable, the following items should be addressed (some of them are already addressed above and are not recalled hereafter – traceability, test coverage, etc.):

- **Item 16:** The perimeter, the depth and the determinism of the usage of inheritance (with the inherent dynamic binding/dispatch effects) and, particularly: the initialisation phase during on time calls, parent and child classes, methods, interfaces, definitions, previously developed SW components, hierarchy respects,

- **Item 17:** The monitoring of the usage of polymorphism.

20 THE USE OF (OCC) OBJECT CODE COVERAGE FOR EQUIVALENCE TO MODIFIED CONDITION DECISION COVERAGE (MCDC)

20.1 BACKGROUND

The use of Object Code Coverage (OCC) may be proposed by some applicants in order to satisfy the ED-12B / DO-178B Modified Condition Decision Coverage (MCDC) objective. MCDC is specified by ED-12B/DO-178B, Table A-7, objective 5, as the degree of test coverage of software structure that should be achieved for Level A software.

Object Code Coverage (OCC) attempts to achieve structural coverage on the object code (i.e. assembly language code) instead of the source code by adhering to strict grammar rules and taking advantage of the "short-circuiting" aspects of modern compilers.

Currently, there is no guidance that would allow the use of OCC as an acceptable alternative to meet the MCDC objective of ED-12B/ DO-178B, Table A-7, and objective 5. This section of this Certification Memorandum is intended to provide the necessary guidance, while meeting the safety objectives, for the use of Assembly Branch Coverage in cases where this approach is selected by an applicant.

20.2 GUIDANCE

EASA has identified the following issues associated with this concept.

When structural coverage analysis at the object (assembly) code level is proposed the applicant should demonstrate that the coverage analysis at the object code level and source code level provide the same level of assurance and addressed in the relevant documentation (i.e. plans, standards and SW life cycle data). The following issues should be addressed (as a minimum):

- The approach should generate the same minimum number of test cases as that needed at the source code level appropriate to the software level of the application (e.g., MC/DC for Level A, decision coverage for Level B).
- The test cases used for coverage should be generated from the requirements (e.g., "structural testing" with module tests based on the code structure should not be used).
- All design and coding rules used to enforce the equivalence should be included in the design and coding standards, verification checklists, etc. and strictly followed.
- Data should be provided to substantiate all assumed compiler behaviour (e.g., short-circuit forms with the appropriate optimization settings). (Note: In some cases, when compilers aggressively optimize or use self-optimization the behaviour becomes unpredictable.)
- Analysis of the object code or qualification of a tool may be necessary to ensure that design and coding rules were followed and that the compiler performed as expected.
- Traceability between object code, source code, design, and requirements should exist.
- Architecture and complexity limitations should be documented and followed (e.g., number of nested *if*s, number of conditions in a decision, nested function calls, etc.).
- The approaches for data coupling analysis and control coupling analysis should be performed by the applicant/developer, whether the coverage is performed on the linked object code or not.
- Data should be available to substantiate any object code not covered.

- The following questions should also be addressed (these are areas that are known to cause problems for many of the object coverage approaches):
  - How are parentheses addressed?
  - How are conditional calls addressed (e.g., jumps in branches)?
  - How are long jump and long throw addressed (do they allow multiple entries and exits)?
  - Are functions limited to only one entry point?
  - Where does transfer of control in the function occur? (Note: It should typically be at the beginning.)
  - How is control from outside to inside a function addressed?
  - How are jump statements addressed (e.g., break, continue, return, goto)?
  - Are bitwise operators used as Boolean operators prohibited?
  - Do functions of the compiler (e.g., pre-parser) need to be qualified for the proposed compiler options and optimizations intended to be used?
  - Is analysis of the object code needed to ensure design and coding rules were followed and that the compiler behaved as expected?
  - Is structural coverage at the appropriate level also achieved for compiler-provided library functions and run-time system-provided library functions used or included in the airborne application?
  - Does the linkage editor (and linking and loading procedures) have the capability to link into the application only those components and functions to be used and for which structural coverage has been achieved, or are unused (dead code) components and functions linked into the application also?
  - Should linker or loader functions be qualified?

21 MERGING HIGH-LEVEL AND LOW-LEVEL REQUIREMENTS

21.1 BACKGROUND

Section 5.0 (second paragraph) of ED-12B / DO-178B states that: "*Software development processes produce one or more levels of software requirements. High-level requirements are produced directly through analysis of system requirements and system architecture. Usually, these high-level requirements are further developed during the software design process, thus producing one or more successive, lower levels of requirements. However, if Source Code is generated directly from high-level requirements, then the high-level requirements are also considered low-level requirements, and the guidelines for low-level requirements also apply*."

This part of section 5.0 addresses the case where the system-level requirements are "directly" highly refined (i.e., created in one refinement step); hence, allowing a single level of software requirements. However, applicants sometimes misuse this paragraph to justify combining high-level requirements (HLR) and low-level requirements (LLR) into the same data item. They establish development cycles that are intended to expand HLR with requirements built during the software design phase. Thus, they are "flattening" the requirements tree and merging all the requirements into a single data item, without establishing traceability between LLR and HLR.

In general, HLR represent "what" is to be designed and LLR represent "how" to carry out the design. Merging these two levels leads to several certification concerns, as described in this section.

21.1.1 ED-12B / DO-178B compliance concerns

Applicants' motivation to expand HLRs and merge them with LLRs during the design phase seems to be to keep all the requirements in a single data item. The resulting data item is the result of a development process that may not comply with ED-12B / DO-178B, since some objectives may not be satisfied. Combining HLRs and LLRs into the same data item is not a recommended practice because process assurance may be affected as follows (as a minimum):

- The complexity of the software requirement document is significantly increased. With no distinction between HLR and LLR, it is difficult to ensure compliance with traceability and verification objectives as defined in DO-178B/ED-12B. Traceability is "the evidence of an association between items, such as between process outputs, between an output and its originating process, or between a requirement and its implementation."
- Demonstration of the correct requirements "management" (particularly for large and complex systems and software) is much more difficult.
- The combination of the "what" (HLR) and "how" (LLR) may impact the relevance of the software life cycle data and the ability to verify the requirements.
- Derived requirements could get missed, which may have safety impacts (since derived requirements should be fed back to the system safety assessment process).
- Compliance demonstration with verification objectives during initial development is more difficult to achieve.
- Compliance demonstration with re-verification objectives for software modifications is more difficult, because change impact analysis will be more complex.
- The ability to manage and analyze the impact of changes/modifications to the airborne software becomes difficult or impossible.
- The consistency of the software requirements document is not ensured when modifying airborne software.
- The consistency and relevance of software requirements document with other development life cycle output data (source code, design architecture, system specification, etc.) is not ensured.
- Transition criteria are difficult or impossible to define and follow when HLR and LLR are combined.

Basically, compliance with DO-178B/ED-12B tables A-4, A-5 and A-6 objectives is weakened when HLR and LLR are merged into a single data item.

For compliance demonstration of DO-178B/ED-12B objectives, applicants should propose further justification and alternate means of compliance to satisfy the applicable objectives and to address the above concerns.

21.1.2 Verification concerns

When airborne software components are large or complex, the software requirements process produces the HLRs and the software design process produces the LLRs and architecture. Thus, HLRs and LLRs are not the outputs of the same development processes (whatever the development process is) and cannot be considered as the same type of requirements (i.e., cannot be HLRs and LLRs at the same time). Therefore, there is no reason to reduce the amount of verification activities required for compliance demonstration with ED-12B / DO-178B section 6, depending on the development cycle followed. In addition, verification activities performed on HLR and LLR cannot be achieved at the same time, since the production processes are distinct.

Therefore, even in case where HLRs and LLRs are merged, the only way to perform verification activities seems to be the same way as it would have been performed in a standard development cycle (distinction maintained between HLRs and LLRs in the software life cycle data).

Furthermore, as documented traceability between HLRs and LLRs is required by ED-12B / DO-178B for verification purposes, loss of traceability is an issue. Alternative methods should be proposed to satisfy the same objectives as the ones satisfied when maintaining documented traceability between HLRs and low-level requirements.

21.1.3 Re-verification concerns (modification of the airborne software)

When LLRs and HLRs are merged into one data item, without traceability between HLRs and LLRs, accurate and applicable re-verification is more difficult. A good change impact analysis is dependent on traceability in order to determine affected components during change. If the HLR to LLR trace is not documented, re-verification becomes difficult or impossible. Therefore:

- the change impact analysis method should establish relevance of the impact perimeter determination and the relevance of verification means (e.g., test subset) chosen to satisfy verification objectives for the modification. If the change impact analysis method is not satisfactory, alternative methods should be proposed to address links between requirements embedded in the software requirements document.
- the level of assurance should be the same level as the one reached when HLRs and LLRs are traceable, maintained, and verified in separate data items.

21.2 GUIDANCE

As described in this Certification Memorandum, EASA does not recommend the merging of HLRs and LLRs into a single data item, because it makes satisfying the objectives of ED-12B / DO-178B difficult or impossible. HLRs state what the system software is to do, and LLRs states how the system software components are to do it. There are different verification approaches and objectives for HLRs and LLRs. For example, many of the HLRs should be verified by the system level and hardware-software integration verification; whereas LLRs typically cannot be verified at that level.

To use section 5 of ED-12B / DO-178B to justify merging of HLRs and LLRs such that visibility of parts of the development process is lost represents a misinterpretation of the original objective of section 5. Combining HLRs and LLRs complicates the ability to demonstrate compliance with ED-12B / DO-178B objectives, introducing confusion and increasing the certification task for software aspects.

Note: There may be some systems where the system-level requirements are "directly" highly refined (i.e., generated in one refinement step). In this case, a single level of software requirements may be feasible, however, if an applicant wishes to use a single level of requirements, they should provide EASA with a rationale to justify this and to show how they will comply with each of the objectives of ED-12B / DO-178B.

ED-12B / DO-178B established objectives to manage and reduce software development complexity, and it recommends separation of development processes and decomposition of associated output data. Merging HLRs and LLRs increases the complexity of software requirements document, software verification process, and software re-verification process. Merging HLRs and LLRs into a single data item for large and complex systems omits some of the objectives and benefits of applying ED-12B / DO-178B (i.e., assurance is lost). Therefore, the practice of combining HLRs and LLRs is not recommended, particularly for large, complex, or safety-critical airborne software. Combining HLRs and LLRs increases the complexity of software development assurance and the risk for undetected errors.

Although, compliance with ED-12B / DO-178B may eventually be shown, when merging HLRs and LLRs, the level of confidence that maintainability will be ensured over a long period of time is not equivalent to the one provided when following the ED-12B / DO-178B objectives. Even if compliance with the ED-12B / DO-178B objectives could be shown, maintainability may not be ensured for the entire life of the product. In reality, if the HLRs and LLRs are merged into a single data item, the impact of such an alternate approach will need to be re-evaluated each time a change is made to the software (to ensure that the objectives of ED-12B / DO-178B section 6 are addressed).

Below is a copy of ED-94B / DO-248B frequently asked question (FAQ) #71. FA Q#71 provides additional insight into the objectives of the ED-12B / DO-178B requirement for traceability.

21.3 EXPLANATION OF THE PURPOSE OF TRACEABILITY FROM ED-94B / DO-248B

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?

Reference: DO-178B/ED-12B: Sections 5.5, 6.2, 6.3, 6.4, 7.2.2, Annex A, and Annex B

Keywords: traceability

Answer:

Traceability is used to:
(1) enable verification of implemented system requirements, high-level requirements, and low-level requirements;
(2) to verify the absence of unintended function and/or undocumented source code; and
(3) to provide visibility to the derived requirements. Traceability applies to both the verification and configuration management processes.

The format for documenting traceability is at the discretion of the applicant or developer. Whilst a matrix format is not required, it has proven successful for many past programs for capturing traceability data in a very usable and concise manner. In some respects, traceability may assist in tracking and controlling the entire software development and its status. Traceability may also assist in establishing the acceptability of derivations of previously developed and approved baselines.

The DO-178B/ED-12B glossary defines traceability as "the evidence of an association between items, such as between process outputs, between an output and its originating process, or between a requirement and its implementation" (reference Annex B). It facilitates evaluation of the processes and their outputs to facilitate the assurance of DO-178B/ED-12B objectives.

Traceability is bi-directional. DO-178B/ED-12B objectives call for both forward (top-down) and backward (bottom-up) traceability. The evidence of traceability may be in a format such that it can demonstrate both forwards and backwards traceability.

Forward traceability shows where each of the requirements, including derived requirements, trace down through the design to their implementation and provides a means to demonstrate that all requirements have been implemented. Forward traceability may also support change analysis by showing those lower-level components (e.g., design, code, test cases) that are affected when a requirement is changed.

Backward traceability shows the origin of the implementation traced up through the design to the requirements. Additionally, the traceability evidence may help demonstrate that nothing has been included in the implementation that is not traceable to the requirements.

To understand traceability, one should consider all aspects of traceability referred to in the following discussion, which lists some of the benefits of traceability and refers to sections of DO-178B/ED-12B that discuss its various objectives, guidance, and activities. Traceability is used in conjunction with the integral processes to provide the following benefits:

- To ensure completeness between development data elements, for example, between system requirements and software requirements, between software requirements (high-level) and software design (low-level), and between software design elements and source code components (reference Section 5.5 of DO-178B/ED-12B).
- To ensure completeness for verification coverage objectives, for example, between software requirements and the verification procedures and verification results for establishing requirements-based test coverage (reference DO-178B/ED-12B Sections 6.2a, 6.2b, 6.3.1f, 6.3.2f, 6.3.4e, 6.4.4.1a, 6.4.4.2.b, and Annex A objectives).
- To facilitate the certification liaison process and certification authority visibility and confidence by providing a usable means for assessing the status and completeness of software life cycle data and its processes throughout the project (for on-site certification authority and designee reviews) and for follow-on certification efforts (reference Sections 9.2 and 12.1.1d of DO-178B/ED-12B).
- To enable the developer to perform change impact analysis; to readily identify software life cycle data that may be affected by changes, both during the development and for post-certification changes; and to determine the scope and required effort to "re-verify" the affected software data items.
- To assist new project personnel implementing software changes to understand the relationships between the software life cycle data during the potentially long service life of many airborne systems and equipment.

With respect to the level of traceability, the developer should select an appropriate level of granularity for which traceability will be provided for each type of element. For example, requirements – one or several functionally related; source code – module or procedure or line of code; verification tests – procedure or case. Since traceability supports efficient access to related elements of different data items, the level of granularity should be selected that best supports effective and efficient access. If the granularity is too coarse, it may be difficult to identify and access all relevant elements efficiently. On the other hand, if the granularity is too fine, maintaining the traceability data itself may become unwieldy and burdensome. Some combination of approaches could be used, for example, using a matrix for general location of related elements and using embedded features such as code comments for correlation at a finer level.

22 CLARIFICATION OF STRUCTURAL COVERAGE ANALYSES OF DATA COUPLING AND CONTROL COUPLING

22.1 BACKGROUND

Numerous misinterpretations exist regarding the purpose of structural coverage analyses of data coupling and control coupling, and acceptable approaches to satisfying EUROCAE/ED-12B / RTCA/DO-178B Objective 8 of Annex A Table A-7. This objective ("Test coverage of software structure (data coupling and control coupling) is achieved") references section 6.4.4.2c of ED-12B / DO-178B, which states that structural coverage "analysis should confirm the data coupling and control coupling between the code components."

This section of this Certification Memorandum discusses the purpose, benefits, challenges, and future concerns of data coupling and control coupling coverage analyses.

Objective 8 of Table A-7 in ED-12B / DO-178B specifies the analysis of data and control coupling for Levels A, B, and C software. ED-12B / DO-178B Annex B defines data coupling and control coupling as follows:

"Data coupling - The dependence of a software component on data not exclusively under the control of that software component."

"Control coupling - The manner or degree by which one software component influences the execution of another software component."

Additionally, ED-12B / DO-178B defines a component as: "A self-contained part, combination of parts, sub-assemblies or units, which performs a distinct function of a system."

Note: ED-94B/ DO-248B, Final Report for Clarification of DO-178B Software Considerations in Airborne Systems and Equipment Certification, Frequently Asked Question (FAQ) #67 provides some clarification on data coupling and control coupling. For reference purposes, the FAQ text is included in this section.

22.2 CLARIFICATIONS

22.2.1 Purpose of data coupling and control coupling analyses

The intent of structural coverage analyses is to provide a measure of the completeness of the testing process of software to ensure that the requirements-based testing (R-BT) of a software program exercised that program's functions and structure adequately to an appropriate level of "completeness" depending on that program's software level and needed integrity. For example, structural coverage analyses of Level C software only needs to provide a measure that all statements were exercised; Level B needs a measure that all statements and all decisions were exercised; and Level A needs a measure that all statements, all decisions and all conditions (plus some independence) were exercised (see Objectives 5, 6 and 7 of Annex A Table A-7). These measurements can be taken and analyzed at the computer program "module" level by reviewing test cases and executing requirements-based tests of that module in isolation from other program modules, and examining, either manually or with a tool, that every statement, decision, and condition (depending on the software level of the module) were exercised, and the module functioned correctly as designed.

"Module" is used in this context to denote a piece or component of the software program rather than the entire program. If an airborne software program consisted of one "module," the above structural coverage would likely be adequate to ensure that that the software program functioned correctly and would not have any side effects leading to anomalous behaviour. However, because of the size and complexity of embedded airborne software programs, having the program consist of a single, self-contained module is neither practical nor good engineering. Thus, developers construct programs of modules or components (both functional and data components) that interact with one another and depend on one another to perform the program's functions.

The intent of the structural coverage analyses of data coupling and control coupling is to provide a measurement and assurance of the correctness of these modules/components' interactions and dependencies. That is, the intent is to show that the software modules/components affect one another in the ways in which the software designer intended and do not affect one another in ways in which they were not intended, thus resulting in unplanned, anomalous, or erroneous behaviour. Typically, the measurements and assurance should be conducted on R-BT of the integrated components (that is, on the final software program build) in order to ensure that the interactions and dependencies are correct, the coverage is complete, and the objective is satisfied.

Satisfaction of this objective is dependent on the detail of the specification of the modules/components' interfaces and the thoroughness of the R-BT for normal range and robustness of the software program. That is, if the interfaces and dependencies are specified in the design requirements, and if those requirements are tested for both normal functioning and robustness, satisfaction of the objective may be a by-product of the design and verification processes. However, if the interfaces and dependencies are not well-specified, and the testing program is minimal, it will be much more difficult to demonstrate the objective has been satisfied.

22.2.2 The intent of objective 8 of table A-7 according to the authors of DO-178B

Several questions about the intent of the structural coverage analysis objective of data coupling and control coupling were posed to the verification group of the committee that produced ED-12B / DO-178B (i.e. the members of SC-167/WG-12 sub-group #4). The majority indicated that the objective was added to address coverage analysis of the testing of the entire software program related to its architecture and the integration of its modules/components. Before the objective was added, the committee recognized that an analysis existed to ensure that all requirements were examined for proper operation by testing of the binary image (i.e., requirements coverage). Additionally, analysis existed to demonstrate that the testing ensured that all of the statements of the source code were executed and for more critical software all the logic statements were thoroughly exercised. However, there was no guarantee that the integration and architecture were fully exercised in the execution environment. Therefore, the data coupling and control coupling objective was added to address this deficit. Committee members claim that the original notion was that call trees/scheduler graphs (control coupling) and data flow diagrams (data coupling) could be annotated (preferably by some tool) with test cases to demonstrate coverage. However, in reality, it turned out to be more complex than this.

To summarize, SC-167/WG-12 sub-group #4 members claimed that the purpose of the structural coverage of the data coupling and control coupling was to evaluate the adequacy of the integrated testing and provide an analysis of the integration activities (e.g., evaluation of call trees, set-use tables, and integration test results). Satisfying the objective occurs during integration and ensures that there is a complete suite of integration tests.

22.2.3 Data and control coupling FAQ from ED-94/ DO-248B

FAQ #67: What are data coupling and control coupling and how are they verified?

Reference: DO-178B/ED-12B: Sections 2.3.1 and 6.4.4.2, Table A-7 of Annex A, and Annex B**.**

Keywords: data coupling; control coupling; verification; coupling

Answer:

The DO-178B/ED-12B glossary (reference Annex B of DO-178B/ED-12B) contains the following definitions:

"Data coupling - The dependence of a software component on data not exclusively under the control of that software component."

"Control coupling - The manner or degree by which one software component influences the execution of another software component."

An example of data coupling is a software component that utilizes parameters with a value that is calculated by a different software component, perhaps being executed at a different iteration rate.

An example of control coupling is a real-time software executive that initiates execution of a software component depending upon external parameters or influences.

Specific testing of data or control coupling is not a DO-178B/ED-12B objective, although this may be an acceptable approach to meet the related coverage objective. DO-178B/ED-12B does have an objective (reference Table A-7, objective 8) that data and control couplings are confirmed (reference Section 6.4.4.2c). This is typically accomplished by review and analysis of test cases and results. Data and control coupling are typically exercised during requirements-based testing (reference Section 6.4.3). Any deficiencies should be addressed by additional verification, as stated in Section 6.4.4.3 of DO-178B/ED-12B.

The verification of data and control coupling depends on the hardware/software interface, software architecture, code structure, and source code language. The verification of data and control coupling involves a combination of:

- Reviews and analysis of Software Architecture, as stated in DO-178B/ED-12B Sections 6.3.3b, 6.3.3c, and 6.3.3f;
- Reviews and analysis of Source Code, as stated in DO-178B/ED-12B Sections 6.3.4b and 6.3.4f; and
- Testing

22.2.4 Design versus integration verification activity

A number of manufacturers perform activities during software design to minimize the data coupling and control coupling issues during integration. This is recognized as a good engineering practice as the Meiler Page-Jones' book entitled *The Practical Guide to Structured Systems Design* (1980) points out. Page-Jones identifies different kinds of coupling to be considered during design (e.g., data coupling, stamp coupling, common coupling, control coupling, and content coupling). In some cases (depending on the architecture), this analysis in the software design phase can be used to supplement the software/software integration activity.

However, the DO-178B/ED-12B objective 8 of Table A-7 is primarily intended to be a verification of the integration activity; that is, verification that the interfaces and dependencies between the software program's modules/components were implemented as designed and are correct. Satisfying the objective is intended to provide a measure of the completeness of integration verification (R-BT of the integrated software program's structure, interfaces, and dependencies).

Many applicants find that documenting the data coupling and control coupling during software design provides the requirements to verify during the software/software integration and hardware/software integration verification process. That is, good documentation of the design helps to satisfy the objective during the integration testing (R-BT coverage of interface and dependency requirements).

22.2.5 EASA perspective on the purpose of data coupling analysis

EASA proposes that the purpose of data coupling analysis is to:

- Be a completion check of the integration testing effort. The analysis also provides insight into the structural robustness of the data structures used by the program. Basically, data coupling analysis is intended to enforce good software engineering practices. Data coupling analysis becomes particularly important when partitioning and other protection means are implemented in the software.

- Identify data dependencies. As an example, a data dependence exists between two components when one component defines a data object and the other component uses the definition of that data object under some operational scenario. In this example, the data user is dependent on the data definer.
- Verify data interfaces between modules/components through testing and analysis (test it, then measure it).
- Identify inappropriate data dependencies.
- Define and evaluate the extent of interface depth.
- Determine and minimize coupling interdependencies.
- Determine and maximize cohesion.
- Evaluate need for and accurate use of global data.
- Evaluate input/output data buffers.
- Bound impact of change and requirements effect(s).

22.2.6 EASA Perspective on the purpose of control coupling analysis

EASA proposes that the purpose of control coupling analysis is to:

- Be a complementary completion check of the integration testing effort (i.e., it complements data coupling analysis). The analysis also provides insight into the structural robustness of the execution, timing, and scheduling. Basically, control coupling is intended to enforce good software engineering practices. Control coupling becomes particularly important when partitioning and other protection means are implemented in software.
- Identify control dependencies. A control dependence exists between two components when the execution of one depends on the other. For example, one module/component calls the other under some operational scenario (i.e., the callee is dependent on the caller). Another example is where one module/component defines the data objects that determine the execution sequence taken by the other module/component under some operational scenario.
- Identify inappropriate control dependencies.
- Verify correct execution call sequence (across modules/components/parts/units/objects).
- Define and evaluate the extent of interface depth.
- Assist in verifying scheduling (e.g., detects problems with call sequences that may cause frame overrun).
- Assist in worst-case execution time (WCET) analysis (a side benefit).
- Bound impact of change and requirements effect(s).

22.3 COMMON BENEFITS AND PROBLEMS WITH APPLYING DATA COUPLING AND CONTROL COUPLING ANALYSES

22.3.1 Benefits of good design and integration practices

Certification authorities have observed that applicants who specify a good design and have well-defined integration practices can identify and address data coupling and control coupling issues, and are able to:

- Provide a better awareness of functionality.
- Reduce the number of test cases needed to cover functionality and the supporting code structure, code interfaces, and requirements.
- Perform more efficient and effective change impact analysis.
- Find errors that are difficult to find in the lab testing and could be costly to fix in the field.
- Perform more effective maintenance.

22.4 Guidance for Satisfying the Data Coupling and Control Coupling Analyses Objective

EASA and other certification authorities have observed a number of problems with the application of data coupling and control coupling analyses to airborne software. The guidance below should be followed in order to proactively address these issues:

- Applicants should address the data coupling and control coupling analyses in their plans (i.e., plan upfront how they will perform these analyses). In some cases, it may be distributed among several plans. However it is documented, it must provide complete and accurate rationale (i.e., it must be thorough).
- Applicants should consider data coupling and control coupling as part of their development/design effort (e.g., specify interface (I/O) requirements and dependencies between components).
- Applicants should substantiate their rationale for "analysis" and/or "testing" aspects of satisfying the objective. The objective may be satisfied as a static activity (e.g., looking at a link map or call tree), a dynamic activity (e.g., running tests), or a combination.
- Applicants should realize that data and control coupling analysis are two different analyses (i.e., they should not be combined into one activity).
- If tools are used, the determination of whether they need to be qualified or not should be evaluated and justified.
- If selective linkers are used, their effect on data coupling and control coupling must be analyzed.

23 THE VALIDATION AND VERIFICATION OF FORMALISED AND MODEL-BASED SOFTWARE REQUIREMENTS AND DESIGNS

23.1 BACKGROUND

Some pieces of aircraft equipment may embed software that was designed by using formalised specification languages such as SAO (Specification Assistée par Ordinateur), or LDS (Langage De Spécification), or by using model-based specifications produced by means of tools such as SCADE or Matlab/Simulink.

Neither the current standard applicable for airborne software (ED-12B / DO-178B) nor the current standard for system development (ED-79 / ARP4754) directly addresses the verification or validation activities that should be carried out when formalized specifications or model-based specifications are used. This section of this Certification Memorandum is intended to provide clarification as to which validation and verification objectives apply in such cases and which verification and validation activities should be conducted.

Some applicants might not have ED-79 / ARP4754 as part of their certification basis. For the paragraphs of this section that refer to activities of ED-79 / ARP4754 to be carried out in order to comply with its objectives, such applicants should state which activities they will conduct that are equivalent to the specific ED-79 / ARP4754 activities that are called for.

23.2 GUIDANCE

23.2.1 Formalized designs, formalized requirements and higher-level requirements

Since this section introduces some new terms that do not exist within ED-12B / DO-178B and that are necessary in order to understand this section, the definitions of the following terms are repeated here for the sake of clarity and to ensure that the guidance contained in these definitions is applied.

Formalized Design - A Formalized Design may be a model produced by the use of a modelling tool or it may be a design stated in a formalized language. In either case, a Formalized Design should contain sufficient detail of such aspects as code structures and data / control flow for Source Code to be produced directly from it, either manually or by the use of a tool, without any further information. A Formalized Design is therefore equivalent in its role to a conventional ED-12B / DO-178B software design, but it is stated in a formalized manner.

Formalized Requirements - Formalized Requirements may be produced by the use of a modelling tool or they may be requirements stated in a formalized language. They contain high level requirements that can later be implemented in either a Formalized Design or in a conventional software design. Formalized Requirements should neither contain Software design nor Software architecture details. A set of Formalized Requirements is therefore equivalent in its role to a set of conventional ED-12B / DO-178B software high-level requirements, but the requirements are instead stated in a formalized manner.

It is not necessary to produce a set of Formalized Requirements in order to produce a Formalized Design, and in most cases, the only formalized part of the software life-cycle data will be a Formalized Design. Formalized Requirements have been included here as a provision for future developments in which they may be used.

Applicants should categorize any formalized specification / model they intend to use as being either a Formalized Design or as a set of Formalized Requirements so that the objectives and activities that apply to them can be clearly determined. (See section 23.2.2 below.) In cases where such a formalized specification or model contains any design details, the formalized specification or model should be categorized by the applicant as a Formalized Design, as this is how EASA will categorize it.

Higher-level Requirements - In order to produce either a set of Formalized Requirements or a Formalized Design, a set of requirements at a higher-level of abstraction is needed in order to capture the requirements for the Formalized Requirements or Formalized Design and to describe what the resulting formalized item should contain. Such requirements are therefore known hereafter in this Certification Memorandum as 'higher-level requirements'. The data item(s) that act as higher-level requirements should be identified during the planning stage, as described in the paragraph immediately below.

Higher-Level Requirements should be developed and stated in a different manner than is used for the Formalised Requirements or Formalized Design that is to be developed from them.

The objective to be fulfilled by producing Higher-Level Requirements is to describe and define the functional and performance requirements that are then either further elaborated into a set of Formalized Requirements or that are directly implemented in a Formalized Design.

23.2.2 The system / software planning process

When an applicant intends to develop any Formalized Requirements or Formalized Designs as part of their processes for producing airborne software, the applicant should, in their software planning process:

a) Clearly state that their system / software life cycle data will include items that fit into the categories of Formalized Requirements and / or Formalized Designs and identify which of their data items fits which of these categories. If the applicant does not use the terms 'Formalized Requirements' and 'Formalized Designs' in their documentation, they should indicate which terms they use for these items.

b) Identify which CSCIs will be wholly or partially developed by the use of which formalized items, which parts of the functionality of those CSCIs those formalized items will represent, and which CSCIs or parts of CSCIs will be developed in a conventional manner.

c) Identify their processes for system development, requirement validation, software development and verification for both their formalized items and for the conventional ED-12B / DO-178B software of their system, clearly identifying and detailing the parts of their process that are common to both formalized and conventional software and the parts that are different.

d) Identify which set of requirements in their set of documentation acts as the Higher Level Requirements for their Formalized Design and/or their Formalized Requirements.

e) If a Formalized Design will be used, identify the design standards with which the Formalized Design should comply.

f) If Formalized Requirements will be used, identify the requirement standards with which the Formalized Requirements should comply, (NOTE – the sets of standards that apply to Formalized Designs and to Formalized Requirements should be different due to their different levels of abstraction and their different content).

g) Identify the requirement and design standards with which components developed using conventional methods should comply and the coding standards with which all the Source Code should comply.

h) Identify which parts of their system and software processes / documentation will demonstrate compliance with each of the objectives of ED-12B / DO-178B for their conventionally-developed components and for their formalized components, given the process clarifications that are provided below, and their processes for compliance with the objectives of ED-79 / ARP4754.

i) State which tools will be used during the development and verification of their conventionally developed software components and which will be used for development and verification of their formalized system / software components, stating which tools will be qualified, and identifying any certification credit sought against ED-12B / DO-178B objectives for the use of qualified tools.

j) Where the tools used include a simulator or emulator, the applicant should identify the differences between the simulator / emulator and the target processor and justify why those differences are acceptable. If the applicant seeks any certification credit against ED-12B / DO-178B objectives for use of a simulator or emulator, then tool qualification may be required, and the applicant should clearly state in their plans which certification credit is sought against which ED-12B / DO-178B objectives and whether the simulator / emulator will be qualified.

23.2.3 Types of system / software life-cycle

Higher-level requirements and the Formalized Requirements and / or Formalized Design that are developed from those requirements may be produced at various stages of the system life-cycle or the software life-cycle. The figure below shows five different types of life-cycle, which life-cycle data items are produced in each life-cycle, which of them are formalized and the system or software processes that produce them.

The shading in the figure below shows the types of life-cycle that have common aspects, as described below, allowing them to be grouped and handled in a similar manner in terms of activities that should be conducted and objectives with which compliance should be shown.

Process that generates the life-cycle data	**Type #1**	**Type #2a**	**Type #2b**	**Type #3a**	**Type #3b**
System Requirement and System Design Processes	System requirements allocated to software	Higher-level requirements	Higher-level requirements	Higher-level requirements	Higher-level requirements
			Formalized Design		
Software Requirement and Software Design Processes	Higher-level requirements (= Software high-level requirements)	Formalized Design		Formalized Requirements (= Higher-Level Requirements for Formalized Design.)	Formalized Requirements
	Formalized Design			Formalized Design	ED-12B / DO-178B Design Description
Software Coding Process	Source Code	Source Code	Source Code	Source Code	Source Code

Figure 1 – Types of System / Software Life-Cycle

For each of their CSCIs that are developed using Formalized Requirements or Formalized Designs, applicants should identify in their plans which of the types of life-cycle described below they intend to use for those formalized items.

In the paragraphs that follow, the artefacts produced and the activities to be conducted on them are mapped to the conventional system and software artefacts and activities described in ED-79 / ARP4754 and ED-12B / DO-178B.

For software related artefacts, the ED-12B / DO-178B objectives that apply and that have to be fulfilled are also listed.

Activities for the review and analysis of requirements at the system level are referred to in ED-79 / ARP4754 as being validation activities but in ED-12B / DO-178B, the review and analysis of requirements at the software level are referred to as being verification activities. This Certification Memorandum will, therefore, use the term 'validation' for these activities at the system level and 'verification' for these activities at the software level.

23.2.4 Type 1 – Formalized design replaces conventional ED-12B / DO-178B software design

Process that generates	**Type #1**

the life-cycle data	
System Requirement and System Design Processes	System requirements allocated to software
Software Requirement and Software Design Processes	Higher-level requirements (= Software high-level requirements)
	Formalized Design
Software Coding Process	Source Code

In this life-cycle, the System Requirements Allocated to Software are elaborated into conventional ED-12B / DO-178B Software High-level Requirements.

The Software High-level Requirements are then used as the higher-level requirements from which a Formalized Design is developed instead of a conventional ED-12B / DO-178B software design.

Source Code is produced directly from the Formalized Design.

This is partly a conventional ED-79 / ARP4754 and ED-12B / DO-178B system and software life-cycle, with the exceptions that a Formalized Design is used instead of a conventional software design and that the Source Code may be produced by an auto-coding tool if such a tool is used, rather than by manual coding.

23.2.4.1 System Requirement Validation

In this life-cycle, the System Requirements Allocated to Software should be validated as described in ED-79 / ARP4754 Section 7 so as to ensure they are complete and correct. The applicant should identify in their System Validation Plan the means they intend to use to validate the system requirements allocated to software, which may include reviews, analysis, simulation or test.

23.2.4.2 Software Requirement / Higher-level Requirement Verification

In this life-cycle, the ED-12B / DO-178B Software High-level Requirements act as the higher-level requirements from which a Formalized Design is developed. These higher-level requirements / Software High-level Requirements should be shown to comply with the System Requirements Allocated to Software as in a conventional ED-12B / DO-178B process and should be reviewed and analyzed as described in ED-12B / DO-178B paragraph 6.3.1 to ensure that they comply with the objectives for Software High-level Requirements shown in ED-12B / DO-178B Table A-3.

23.2.4.3 Formalized Design Verification Activities

ED-12B / DO-178B verification activities should be conducted on the Formalized Design produced in this life-cycle as described in the sections below with the following headings -

- Verification of Formalized Designs
- Coverage of Formalized Designs.

Simulation may be used as part of these verification activities, in which case the text in the following section also applies –

- Simulation of Executable Formalized Designs.

23.2.4.4 *Verification of the Executable Object Code*

The Executable Object Code (EOC) should be shown to comply with the objectives in ED-12B / DO-178B Table A-6, including compliance with the software high-level requirements (which are the higher-level requirements for the Formalized Design) and compliance with the low-level requirements, which are within the Formalized Design.

Requirement-based test cases and procedures to be executed against the EOC should cover normal range and robustness values, the equivalence classes of the input data and any potential singularities.

The ED-12B / DO-178B Hardware / Software Integration testing as described in ED-12B / DO-178B paragraph 6.4.3 a. must be conducted with the ED-12B / DO-178B Executable Object Code loaded onto the target processor in the host environment.

23.2.4.5 *Other Required Activities*

The other general activities and objectives that are applicable to this life-cycle are shown below in the section dealing with General Principles and Activities.

These include –

- Traceability and Granularity of Requirements / Design Elements.
- Derived Requirements / Elements.
- Non-Functional Requirements.
- Requirement Coverage Analysis.
- Verification that Source Code Complies with Requirements and Standards.
- Structural Coverage of Source / Object Code.
- Qualification of Auto-coding Tools.
- Compliance with Standards.
- Independence.

23.2.4.6 *System Requirement Verification*

Once the testing of the software and airborne electronic hardware components of a system has been completed, the complete set of system level requirements needs to be verified to the degree required by section 8 of ED-79 / ARP4754 for the DAL of the system. The verification methods and activities to be used for system level verification should be described in the System Verification Plan and conducted in accordance with that plan. The results should be recorded in the System Verification Data.

23.2.5 Types 2a and 2b – Formalized design replaces software high-level requirements and software design

Process that generates the life-cycle data	**Type #2a**	**Type #2b**
System Requirement and System Design Processes	Higher-level requirements	Higher-level requirements Formalized Design
Software Requirement and Software Design Processes	Formalized Design	
Software Coding Process	Source Code	Source Code

In these types of life-cycle, the system development processes produce a set of higher-level requirements that are developed into a Formalized Design, either by the system development processes or by the software development processes.

Source Code is produced directly from the Formalized Design, either manually or automatically.

These life-cycles differ considerably from a conventional ED-79 / ARP4754 and ED-12B / DO-178B system and software life-cycle, as conventional software high-level requirements and a conventional software design are both replaced by a Formalized Design. An auto-coding tool may be used to produce the Source Code automatically rather than by manual coding.

23.2.5.1 Validation of Higher-level Requirements

In these life-cycles, the higher-level requirements from which the Formalized Design is developed are at the system level. These higher-level requirements at the system level should be validated in the manner described in ED-79 / ARP4754 Section 7 so as to ensure that they are complete and correct. The applicant should identify in their System Validation Plan the means they intend to use to validate the system level requirements allocated to software, which may include reviews, analysis, simulation or test.

23.2.5.2 Formalized Design Verification Activities.

ED-12B / DO-178B verification activities should be conducted on the Formalized Design produced in this life-cycle as described in the sections below with the following headings -

- Verification of Formalized Designs.
- Coverage of Formalized Designs.

Simulation may be used as part of these verification activities, in which case the text in the following section also applies –

- Simulation of Executable Formalized Designs.

23.2.5.3 Verification of the Executable Object Code

The Executable Object Code (EOC) should be shown to comply with the objectives in ED-12B / DO-178B Table A-6, including compliance with the higher-level requirements (which take the place of the software high-level requirements in Table A-6) and compliance with the requirements contained within the Formalized Design (which take the place of the software low-level requirements in Table A-6).

Requirement-based test cases and procedures to be executed against the EOC should cover normal range and robustness values, the equivalence classes of the input data and any potential singularities.

The ED-12B / DO-178B Hardware / Software Integration testing as described in ED-12B / DO-178B paragraph 6.4.3 a. must be conducted with the ED-12B / DO-178B Executable Object Code loaded onto the target processor in the host environment.

23.2.5.4 Other Required Activities.

The other general activities and objectives that are applicable to this life-cycle are shown below in the section dealing with General Principles and Activities.

These include –

- Traceability and Granularity of Requirements / Design Elements.
- Derived Requirements / Elements.
- Non-Functional Requirements.
- Requirement Coverage Analysis.
- Verification that Source Code Complies with Requirements and Standards.
- Structural Coverage of Source / Object Code.
- Qualification of Auto-coding Tools.
- Compliance with Standards.
- Independence.

23.2.5.5 System Requirement Verification.

Once the testing of the software and airborne electronic hardware components of a system has been completed, the complete set of system level requirements need to be verified to the degree required by section 8 of ED-79 / ARP4754 for the DAL of the system. The verification methods and activities to be used for system level verification should be described in the System Verification Plan and conducted in accordance with that plan. The results should be recorded in the System Verification Data.

23.2.6 Types 3a and 3b - Formalized requirements replace software high-level requirements

Process that generates the life-cycle data	Type #3a	Type #3b
System Requirement and System Design Processes	Higher-level requirements	Higher-level requirements
Software Requirement and Software Design Processes	Formalized Requirements (= Higher-Level Requirements for Formalized Design.)	Formalized Requirements
	Formalized Design	ED-12B / DO-178B Design Description
Software Coding Process	Source Code	Source Code

In these types of life-cycle, Formalized Requirements are developed from system level higher-level requirements and are used in place of conventional ED-12B / DO-178B software high-level requirements. These Formalized Requirements may then be further developed into either a Formalized Design or into a conventional ED-12B / DO-178B software design.

Source Code is then produced from the Formalized Design / the software design.

These life-cycles differ from a conventional life-cycle involving both ED-79 / ARP4754 and ED-12B / DO-178B in that the software high-level requirements are replaced by a set of Formalized Requirements, and in the first case, a conventional software design is replaced by a Formalized Design. In the case where a Formalized Design is produced, an auto-coding tool may be used to automatically produce the Source Code rather than by producing it by means of manual coding.

23.2.6.1 *Validation of Higher-level Requirements*

In these cases, the higher-level requirements from which the Formalized Requirements are developed are at the system level. These higher-level requirements at the system level should be validated in the manner described in ED-79 / ARP4754 Section 7 so as to ensure that they are complete and correct. The applicant should identify in their System Verification Plan the means they intend to use to validate the system level requirements allocated to software, which may include reviews, analysis, simulation or test.

23.2.6.2 *Verification of Formalized Requirements.*

The Formalized Requirements produced in these life-cycles should be shown to comply with their higher-level requirements (which are at the system level) by conducting activities to show compliance with the objectives for software high-level requirements shown in ED-12B / DO-178B Table A-3 as in a conventional ED-12B / DO-178B process. The Formalized Requirements should, therefore, be reviewed and analyzed as described in ED-12B / DO-178B paragraph 6.3.1.

If a Formalized Design is developed from the Formalized Requirements, then the Formalized Requirements serve as the higher-level requirements for the Formalized Design, and these higher-level requirements are verified by verifying the Formalized Requirements as described above. Testing conducted in a simulation of the Formalized Design (as described below in the paragraph on Simulation of Executable Formalized Designs) may be used to provide partial evidence that the Formalized Requirements were complete and correct.

23.2.6.3 Formalized Design Verification Activities (where applicable).

If the life-cycle includes a Formalized Design, then ED-12B / DO-178B verification activities should be conducted on that Formalized Design.

The activities for the verification of Formalized Designs are described in the paragraphs below that have the following headings –

- Verification of Formalized Designs.
- Coverage of Formalized Designs.

Simulation may be used as part of these verification activities, in which case the text in the following section also applies –

- Simulation of Executable Formalized Designs.

23.2.6.4 Verification of a Conventional ED-12B / DO-178B Design (in Type 3b).

If a conventional ED-12B / DO-178B software design is produced, then that software design should be shown to comply with the Formalized Requirements by conducting the review and analysis activities described in ED-12B / DO-178B paragraphs 6.3.2 and 6.3.3 so as to comply with the objectives of Table A-4.

23.2.6.5 Verification of the Executable Object Code.

The Executable Object Code (EOC) should be shown to comply with the objectives in ED-12B / DO-178B Table A-6. This includes compliance with the Formalized Requirements (which take the place of the software high-level requirements in Table A-6). For Type 3a, this also includes compliance with the requirements contained within the Formalized Design (which take the place of the software low-level requirements in Table A-6). For Type 3b, this also includes compliance with the software low-level requirements within the conventional DO-178B software design.

Requirement-based test cases and procedures to be executed against the EOC should cover normal range and robustness values, the equivalence classes of the input data and any potential singularities.

In all cases, the ED-12B / DO-178B Hardware / Software Integration testing as described in ED-12B / DO-178B paragraph 6.4.3 a. must be conducted with the ED-12B / DO-178B Executable Object Code loaded onto the target processor in the host environment.

23.2.6.6 Other Required Activities.

The other general activities and objectives that are applicable to this life-cycle are shown below in the section dealing with General Principles and Activities.

These include –

- Traceability and Granularity of Requirements / Design Elements.
- Derived Requirements / Elements.
- Non-Functional Requirements
- Requirement Coverage Analysis.
- Verification that Source Code Complies with Requirements and Standards.
- Structural Coverage of Source / Object Code.
- Qualification of Auto-coding Tools.
- Compliance with Standards.
- Independence.

23.2.6.7 System Requirement Verification.

Once the testing of the software and airborne electronic hardware components of a system has been completed, the complete set of system level requirements need to be verified to the degree required by section 8 of ED-79 / ARP4754 for the DAL of the system. The verification methods and activities to be used for system level verification should be described in the System Verification Plan and conducted in accordance with that plan. The results should be recorded in the System Verification Data.

23.2.7 Verification of formalized designs

In the types of life-cycle described above where Formalized Designs are produced, those designs take the place of conventional ED-12B / DO-178B software designs. Activities should be identified and conducted by each applicant to show that each Formalized Design is complete and correct, that it complies with its higher-level requirements and that it does not include unintended functionality.

In ED-12B / DO-178B, these activities are described as verification activities. The applicant should conduct reviews and analyses of their Formalized Designs in compliance with ED-12B / DO-178B paragraphs 6.3.2 and 6.3.3 so as to show compliance of each Formalized Design with the objectives in Table A-4 of ED-12B / DO-178B.

In these cases, the Formalized Design is taken to contain the low-level requirements and the architecture.

An additional means to show the compliance of a Formalized Design with these ED-12B / DO-178B objectives may be the use of simulation, as described below.

23.2.8 Simulation of executable formalized designs

Executing a simulation of a Formalized Design may allow problems with the design and possibly with its higher-level requirements to be detected early in the software development process and to be corrected even before the production of the Source Code and the testing of the Executable Object Code.

Where the tools used to produce a Formalized Design permit that design to be executed within a simulator, simulation cases and procedures may be developed against the higher-level requirements in order to support the verification of the Formalized Design (cf. 23.2.7). To this purpose, the guidance in section 23.2.8.1 below should be followed.

Under certain conditions and with proper planning, it may also be possible to take some credit from simulation activities to support the verification of the Executable Object Code. To this purpose, the guidance in section 23.2.8.2 should be followed. However, as simulation involves a different environment and potentially different source code or executable object code from those used when testing the Source Code and Executable Object Code on the target processor, it is not possible to claim credit for the use of simulation instead of conducting Hardware/Software integration testing activities.

23.2.8.1 Simulation for verification of the Formalized Designs (reviews and analyses)

Simulation of the Formalized Design against the higher-level requirements can provide some evidence as to whether the Formalized Design complies with the higher-level requirements, which can be used to show compliance with some of the objectives in ED-12B / DO-178B Table A-4 (typically objectives 1, 2, 4, 7, 8, 9, 11). Other objectives such as compatibility with the target computer, conformance to standards or partitioning integrity cannot be demonstrated through simulation and should be fulfilled by means of conventional reviews and analyses.

To this purpose, the applicant should:

a. Determine precisely in the PSAC which objectives related to reviews and analyses are planned to be covered by simulation and justify how the identified simulation activity fulfils those objectives.

b. Ensure that the simulation cases and procedures are developed and reviewed according to the guidance in section 23.2.8.3.

23.2.8.2 Simulation for verification of the Executable Object Code (testing)

It may be possible to take some credit from executing a simulation of the Formalized Design in order to support compliance with objectives 1 and 2 of ED-12B / DO-178B Table A-6 and objective 3 of Table A-7.

Note: As the Formalized Design cannot be used to verify itself, compliance with objectives 3 and 4 of Table A-6 cannot be wholly or partly claimed based on the use of simulation of the Formalized Design. Similarly, compliance with objective 5 of Table A-6 cannot be wholly or partly claimed based on the use of simulation of the Formalized Design because this objective is related to hardware compatibility aspects.

To this purpose, the applicant should:

a. Ensure that the Formalized Design used for simulation is identical to the one used to produce the Source Code.

b. Determine precisely in the PSAC which testing objectives are planned to be covered by simulation and justify how the identified simulation activity fulfils those objectives.

c. Perform an analysis to identify any differences between the target environment and the simulation environment and provide a rationale for why these differences are acceptable.

d. Ensure that there are no differences between the source code used for simulation and the Source Code of the final software product.

e. Perform an analysis to identify any differences between the executable object code used for simulation and the Executable Object Code of the final Software product. These differences should be justified and a rationale provided for why they are acceptable.

f. Ensure that the simulation cases and procedures are developed and reviewed according to the guidance in section 23.2.8.3.

For Hardware/Software integration testing (ED-12B / DO-178B section 6.4.3.a), the test procedures must be executed with the Executable Object Code of the final Software product loaded into the target hardware and therefore no credit can be taken for the use of simulation instead of Hardware / Software integration testing.

23.2.8.3 Considerations on simulation cases, procedures and results

In order to gain any certification credit as described in 23.2.8.1 and 23.2.8.2, the simulation cases and procedures should be reviewed against the higher-level requirements. In particular, the applicant should perform analyses to verify that:

- The simulation cases are correct.
- The simulation cases satisfy the criteria of normal range and robustness as defined in section 6.4.2.of ED-12B / DO-178B.
- Simulation cases exist for each requirement that is intended to be verified by simulation.
- The simulation cases were accurately developed into simulation procedures and expected results.
- The simulation results are correct and that discrepancies between actual and expected results are explained.

The same cases and procedures that have been formally reviewed and corrected may be later re-used, if desired, as the basis for the test cases and procedures that are used to test the Executable Object Code against the higher-level requirements (cf. sections 23.2.4.4, 23.2.5.3 and 23.2.6.5).

23.2.9 Coverage of formalized designs

The objective of showing coverage of a Formalized Design is to ensure that it contains no unintended functionality.

It should be demonstrated that full coverage of the functions of each Formalized Design is obtained by executing test cases and procedures that are based on the higher-level requirements.

Coverage of derived low-level requirements in a Formalized Design may be obtained by the use of test cases based on the Formalized Design itself if full coverage of those aspects cannot be obtained by use of test cases based on the higher-level requirements.

Coverage of a Formalized Design may be shown by testing based on the formal verification cases and procedures that are used for ED-12B / DO-178B verification or by the use of simulation test cases and procedures.

The following criteria may be used to assess the coverage of the Formalized Design:

- All the conditions of the logic components.
- All equivalence classes (valid/in-range and invalid/out-of-range classes) and singular points of the functional components and algorithms.
- All transitions of the state machines.
- Coverage of all characteristics of the functionality in context (e.g. Watchdog function is triggered).

The applicant should identify in their Software Verification Plan how they intend to demonstrate coverage of any Formalized Design that they produce.

Deficiencies in the coverage of a Formalized Design should be resolved by actions such as augmentation of the test cases, modification of the higher-level requirements, identification of deactivated elements of the Formalized Design, and the removal of any unintended functionality.

23.2.10 General principles and activities

The following activities apply to any of the cases where either Formalized Requirements or a Formalized Design is used.

23.2.10.1 Traceability and Granularity of Requirements / Design Elements.

In any of these cases where Formalized Requirements and /or Formalized Designs are used, traceability must be shown and maintained from the system level of requirements through each of the successive levels of requirements and / or design to the Source Code. The individual requirements and the design functions of each level need to be identifiable so that they can be traced.

At each level of requirements or design, the requirements or the formalized designs should be expressed with a level of granularity that permits the next level of requirements / design / Source Code to be produced from them, and that permits the requirements / elements / Source Code of the next level to be traced to the requirements or to the design functions from which they were developed.

23.2.10.2 Derived Requirements / Elements.

Derived requirements / derived design elements may be introduced during the development of requirements and designs, whether or not the requirements or designs are formalized. Any derived requirements introduced at any level of requirements or designs should be identified as derived requirements / derived design elements and should be provided to the safety assessment process so they can be assessed for any potential impact on safety. Any impact on safety due to a derived requirement / design element may result in the alteration or deletion of that derived requirement / design element, or the introduction of some form of mitigation.

23.2.10.3 Non-Functional Requirements or Elements.

There may be elements of Formalized Requirements or Formalized Designs that are non-functional. These should be identified, since non-functional elements do not need to be covered by testing or by traceability. It must be ensured that no functional requirement / element traces to a non-functional requirement / element. Non-functional elements should still be reviewed to ensure that they are pertinent to the functional requirements and that they are consistent with them.

23.2.10.4 Requirement Coverage Analysis.

Analysis should be performed to ensure that all requirements at all levels of the system and software are covered by test cases and procedures, whether or not the requirements are developed and captured in a conventional manner, and the test cases and procedures should be augmented if any requirements are not covered by tests.

Coverage of requirements at the software level shows compliance with ED-12B / DO-178B objectives A-7.3 and A-7.4. Coverage of requirements at the system level shows compliance with ARP4754 paragraph 8.4.2.2.

NOTE - If a test case and its corresponding test procedure are developed and executed for higher-level requirement-based testing and satisfy the objectives for coverage of the requirements and of the Formalized Design as well as the structural coverage objectives, it is not necessary to duplicate the tests at the Formalized Design level. The use of tests at the Formalized Design level instead of tests based on higher-level requirements may be less effective because less of the overall functionality in context will be tested.

23.2.10.5 Verification that Source Code Complies with Requirements and Standards.

In any of the above cases, the Source Code should be shown to comply with the objectives in ED-12B / DO-178B Table A-5. In the case of a Formalized Design, the low-level requirements and the architecture with which the Source Code has to comply are contained in the Formalized Design. The Source Code should be reviewed against both the Formalized Design and the Coding Standards and analyzed as described in ED-12B / DO-178B paragraph 6.3.4 in order to comply with the objectives of Table A-5.

23.2.10.6 Structural Coverage of Source / Object Code.

In all these life-cycles described above, structural coverage of the Source Code as described in ED-12B / DO-178B paragraph 6.4.4.2 and Table A-7 of ED-12B / DO-178B should be demonstrated to the degree required for the DAL of the software.

Applicants should identify in their Software Verification Plan the means they intend to use in order to demonstrate structural coverage.

Any deficiencies in structural coverage should be resolved as described in ED-12B / DO-178B paragraph 6.4.4.3.

23.2.10.7 *Qualification of Auto-coding Tools.*

In some cases, the software developer may develop or utilize an auto-coding tool so as to produce code directly from a Formalized Design without the need for manual coding of the elements of the Formalized Design. (Manual coding is often necessary for other parts of the Source Code with which the auto-code has to interface.)

If the software developer wishes to take certification credit against any of the ED-12B / DO-178B objectives due to the use of auto-coding tools, the auto-coding tool will need to be qualified as a development tool as described in ED-12B / DO-178B section 12, and the tool qualification will have to be documented and verified as described in ED-12B / DO-178B section 12.

In addition, if a library of model elements is used to produce Formalized Designs, each of these elements should be developed and verified according to ED-12B / DO-178B guidance at the DAL at which it is intended to be used.

Where certification credit due to the use of an auto-coding tool is sought for the ED-12B / DO-178B verification objectives related to structural coverage analysis, the auto-coding tool will also need to be qualified as a software development tool in the context of the target processor and of the tools used to produce the Source Code.

As part of the qualification of an auto-coding tool, the actual operational context in which the tool is used needs to be taken into account. The aspects to consider may include, but are not limited to, the following:

- The definition of a set of representative input files that include, in particular –
    - All the library elements used to produce Formalized Designs in accordance with the applicable standards.
    - Representative combinations of those library elements.
    - Limits imposed by the applicable standards.
    - The degree of complexity that is permitted for Formalized Designs.
- The execution of the auto-code generator on the set of input files and the generation of Source Code.
- The generation of the Executable Object Code by using the same generation environment as is used for the airborne software, including the same compiler/linker with the same selected options.
- The verification of the compliance of the Executable Object Code with respect to the representative input files.

If the tool used to produce the Source Code from the Formalized Design is not qualified as a ED-12B / DO-178B development tool, no certification credit will be granted for the use of that tool against any of the ED-12B / DO-178B objectives for the verification of the Source Code by review, analysis or test, and the activities corresponding to those objectives should be fully conducted as in a conventional ED-12B / DO-178B software development.

23.2.10.8 *Compliance with Standards.*

All the formalized and conventional life-cycle data items that capture requirements, designs or Source Code should be shown to comply with the standards that are applicable to them and that were identified and documented during the system and software Planning Processes.

According to the level of each item, the standards applicable may be System Requirement Standards, System Design Standards, Software Requirement Standards, Software Design Standards, Software Coding Standards, Formalized Requirement Standards, or Formalized Design Standards. Each standard to be used should be identified and provided in the system / software plans.

23.2.10.9 Independence

Independence in software activities should be ensured in compliance with ED-12B / DO-178B.

Independence in system level activities should be ensured in compliance with the intent of ED-79 / ARP4754 section 7: "The validation process at each level of the requirements hierarchy should involve all relevant technical disciplines, including the safety assessment process"

24 THE USE OF PSEUDO-CODE AS LOW-LEVEL REQUIREMENTS

24.1 BACKGROUND

EASA wishes to discourage the use of pseudo-code as a statement of low-level requirements.

Some applicants have chosen not to carry out the normal ED-12B / DO-178B cycle of developing high-level requirements, low-level requirements and then source code with separate software life-cycle data items for each. Some of them have instead chosen to design their source code in the form of pseudo-code that obviously specifies the structure of the source code that is to be implemented from it. These applicants claim that their pseudo-code is the statement of their low-level requirements and of their functional requirements and some even claim that the pseudo-code design of each code module constitutes a single low-level requirement.

This runs contrary to the normal practice of requirement development, in which there is a gradual elaboration of requirements, such that each successive level of requirements contains more requirements than the previous level, and the requirements are at successively lower levels of abstraction. The pattern of requirements in a pseudo-code design as described above is actually the reverse of a requirement elaboration process and may even be understood as a reverse engineering process. This indicates that requirement information is gradually being absorbed or hidden in the successively lower levels of requirements in a pseudo-code design, which is contrary to the normal practice of ED-12B / DO-178B software life-cycles and prevents requirements being properly traced and tested.

If the normal cycle is followed of developing high-level requirements, more-detailed low-level requirements and then source code, there is no need to produce pseudo-code and it serves no useful function in an ED-12B / DO-178B project.

24.2 PROBLEMS WITH THE USE OF PSEUDO-CODE

EASA considers that the use of pseudo-code to express low-level requirements is not compatible with the ED-12B / DO-178B definition of low-level requirements. Per ED-12B/DO-178B, low-level requirements are requirements from which source code can be developed without additional information. Pseudo-code is developed in order to actually be a specification of the structure and the contents of the source code. The source code is then deliberately almost identical to the pseudo-code, so the source code cannot be said to be developed from the corresponding pseudo-code, except in the sense that it may be stated slightly differently in a compilable language. In such a case, there are no low-level requirements that are actually stated as requirements and all that is provided instead is an implementation of some un-stated low-level requirements in pseudo-code. The source code cannot then be said to be developed from any low-level requirements in such a case because no actual low-level requirements are developed or stated. Consequently, there are no conventional low-level requirements to review, to trace or to test.

The steps of reviewing or testing source code against low-level requirements that are supposedly expressed in pseudo-code, to which the source code is almost identical, cannot therefore be as effective as a review of the source code against conventional low-level requirements and can only produce limited results.

This also means that the step of conducting structural coverage analysis of the source code by executing tests based on low-level requirements that are supposedly stated in pseudo-code is an ineffective activity because tests based on code modules are tests of the structure of the code (which is a kind of structural testing) and are not tests of real requirements. Module tests that are written to drive all the decisions and variables of a code module, based on knowledge of the structure of the code module in either pseudo-code or the source code, will naturally produce complete coverage of the module. Such apparently complete coverage would be obtained even though parts of the module or the entire module might not actually be executed or reachable when the Executable Object Code is run due to the values of variables being changed during execution of the rest of the code, resulting in some decision paths not being taken. If such module tests were to be misinterpreted as being structural coverage tests then the result should always be 100% structural coverage, which means that the activity would not be effective in detecting unintended functionality or unexpected behaviour.

Tests of modules of code, i.e. structural testing, were required in ED-12A / DO-178A, but structural testing is not part of ED-12B / DO-178B and is not equivalent to ED-12B / DO-178B structural coverage analysis or the testing of high and low-level requirements. This distinction is described in depth in ED-94 / DO-248B FAQs #43 and #44.

24.3 CAN TRACEABILITY COMPENSATE FOR NON-PRODUCTIVE STRUCTURAL COVERAGE ANALYSIS?

Software developers using pseudo-code as low-level requirements often claim that they can ensure that the contents of the pseudo-code and source code modules are correct by reviewing them against the high-level requirements and ensuring traceability, even though they are actually performing structural testing rather than structural coverage analysis. Although traceability is a valuable activity in ensuring that the functionality of requirements is fulfilled and that only that functionality is fulfilled, the argument that traceability compensates for the absence of proper structural coverage analysis incorporates several problems.

Firstly, such an argument means that pseudo-code (and therefore the source code) has been developed directly from the high-level requirements, which is not a recommended practice, as shown by the contents of section 21 of this Certification Memorandum.

Secondly, the absence of real low-level requirements means that there is no review of real low-level requirements against the high-level requirements and there is no real review of the source code against real low-level requirements. This is a short-cut of the ED-12B / DO-178B review activities and of the opportunities to discover and remove errors that those reviews provide.

Thirdly, even if the normal stages of review of the low-level requirements and of the source code are conducted in a normal development life-cycle, there may still be dead code, deactivated code or unintended functionality or unexpected behaviour remaining in the source code because review activities are rarely conducted with 100% effectiveness due to human error in the activities. Structural coverage analysis was included in ED-12B / DO-178B as a complementary activity that can catch problems in the source code that were missed during the review activities, or that might not have been readily apparent due to the complexity of data and control flow. It is therefore important that proper structural coverage analysis against high-level requirements and real low-level requirements is conducted in addition to the reviewing activities called for in ED-12B / DO-178B. The reviewing and traceability activities of ED-12B / DO-178B are therefore not a substitute for structural coverage analysis, especially when the reviewing and traceability activities are not properly conducted, which is the case when pseudo-code is used to express so-called low-level requirements, as mentioned in the second item above.

24.4 GUIDANCE

EASA considers that when pseudo-code is used instead of real low-level requirements from which code can be developed, this prevents the usual ED-12B / DO-178B review and traceability activities on low-level requirements and source code from being conducted as required and it prevents meaningful structural coverage analysis from being conducted, as a form of structural testing is conducted instead. Structural testing is not required in ED-12B / DO-178B and it cannot be used to replace the structural coverage analysis activities of ED-12B / DO-178B. Traceability cannot be used to compensate for the lack of structural coverage analysis.

EASA therefore considers that -

1 - The use of pseudo-code alone to express ED-12B / DO-178B low-level requirements is neither required nor useful and that developers of airborne software should not use pseudo-code alone to express low-level requirements and should not incorporate it in their software life-cycle data. Source code should be developed from low-level requirements that are not stated in the form of pseudo-code and that do not specify the structure of the source code.

2 – When pseudo-code is used in low-level requirement development in addition to other life cycle data (e.g. a data dictionary, real low-level requirements, etc.), some issues still remain. In some cases, the use of a kind of pseudo-code may ease the understanding of the flow of the low-level requirements (e.g. where there is an if-then-else structure). In such cases, the developer should perform:

- An analysis against the high-level requirements showing that none of the pseudo code included in the low-level requirements introduces any unintended functionality or unexpected behaviour.
- An analysis that all the verification cases cover actual low-level requirements and that they do not cover the structural form of the pseudo-code.
- An analysis showing that the activities defined in ED-12B / DO-178B section 6.4.3.c have not been affected by the use of pseudo-code (e.g. due to the use of implicit else statements).
- An analysis that the structural coverage analysis defined in ED-12B / DO-178B section 6.4.4.3 has not been affected by the use of pseudo-code (due to conducting structural testing vs. requirement-based testing).

In cases where pseudo code structures are expressed using semantic constructions that are not common in spoken English (e.g. case, default, until, etc.), this would not be considered as an aid to understanding the low-level requirements. This would be considered to be too close to real source code and therefore, instead, to fall into the category of bullet 1 above.

25 STACK OVERFLOWS

25.1 PURPOSE

The purpose of this Section is to provide guidance for determining whether software designs should implement protection mechanisms to cope with stack overflows.

ED-12B / DO-178B already provides guidance regarding stack usage and stack overflows in the areas of code reviews and requirement based verification, however, it does not cover the possibility that a data corruption may occur and lead to a stack overflow.

The purpose of this section is not to change the intent of ED-12B / DO-178B but rather to provide details of specific aspects of stack usage and stack overflows to which attention should be paid. Although this section only refers to stack overflows, it should be understood that stack underflows also need to be taken into account in each case where stack overflows are mentioned and that the guidance in this section also applies to any other area of dynamic memory that is used during the runtime of an airborne software program, such as a heap.

25.2 BACKGROUND

Most avionics systems incorporate a real-time operating system, which may have many stacks that are used dynamically. In addition, there may be secondary stacks which may be used to manage aggregate data (for example dynamically sized arrays) or data whose lifetime extends beyond its calling frame (e.g. a locally declared record being returned from the function where it is declared).

Depending on the programming language used and the implementation of the compiler, various problems may arise in the event of a stack overflow. The problems may manifest themselves as:

1. A corruption of data (if the stack overflow causes the software to write into a data location).
2. A machine exception (if the stack overflow is adjacent to protected memory, either code or read-only memory).
3. An unpredictable program execution (if the corruption of data occurs and code addresses are corrupted, e.g. the return address of a function).
4. A software exception (if the program contains code to check the stack).

As secondary stacks are used less frequently and their use is typically under the control of run-time functions, stack checks are often put in place to monitor all secondary stack usage. Typically, run-time checks on primary stacks are avoided for performance reasons.

ED-12B/DO-178B requests that stack usage should be addressed appropriately by review of the source code and stack overflows should be addressed by requirement-based hardware/software integration testing.

Analysis of the theoretical worst case scenario helps to determine whether the use and implementation of stacks has been well-designed in order to adequately manage data that needs to be handled through stack usage.

Static analysis of worst case stack use consists of identifying all the functions in the program and obtaining their stack frame sizes, determining the call graph of each separate execution thread (typically a task), and combining this data into a set of worst case stack sizes for each stack. This can be done at the source code level by counting the sizes of all data declarations and parameters. This raises the problems that the sizes chosen by the compiler, any alignment gaps and any temporary compiler-allocated data on the stack would all need to be accounted for.

Stack analysis may also be performed by testing. The typical approach is to fill the memory with a certain memory pattern and to execute tests which force the maximum usage of the stack. As this behaviour is dynamic, it may be difficult to determine the worst-case scenario except for simple programs.

Nonetheless, there is an additional potential risk that an unintended stack overflow may occur during the execution of a program for various reasons, such as:

- A hardware failure.
- A software development error.
- An unintended software behaviour.
- A memory corruption.
- A single event upset (SEU).
- etc.

In such cases, when a stack overflow occurs during the execution of level A software, the consequences could be catastrophic as the software may be out of control.

Although an SEU is unlikely to be repeated, if a stack corruption or an overflow occurs due to a software error, then given the same program state and input conditions, the software error will be repeated. Such a repetition may cause the recovery mechanism to be ineffective.

25.3 GUIDANCE

EASA considers that for Level A software, there is a need to consider the possibility of a stack overflow occurring in flight. This means that theoretical measurements performed to determine that stack overflows cannot occur may be not sufficient to ensure that the software behaves as intended. Consequently, stack monitoring may be necessary to detect any stack overflows and handle the potential risks.

To conduct run time monitoring of stack overflow is not trivial and usually implies the incorporation of specific requirements to cover those concerns. The following activities have been identified:

a) An analysis should be performed to define whether or not it is necessary to perform continuous stack monitoring, based on criteria such as the software criticality, the stack type, the use of built-in monitors, etc.),
b) If continuous monitoring of the stack is used, it should be performed in real time,
c) The monitoring mechanism (e.g. the monitoring of the stack pointer) should be specified in the requirements,
d) In the event that the monitor detects an overflow, the expected behaviour (e.g. exception) should be specified and verified accordingly,

There may be several stacks with different monitoring policies (data, executive-level, program counters, etc.) and the above analyses may be needed for each stack.

26 REMARKS

1. This EASA Proposed Certification Memorandum will be closed for public consultation on the **25th of March 2011**.

2. Comments regarding this EASA Proposed Certification Memorandum should be referred to the Certification Policy and Planning Department, Certification Directorate, EASA. E-mail CM@easa.europa.eu or fax +49 (0) 221 89990 4459.

3. For any questions concerning the technical content of this EASA Proposed Certification Memorandum, please refer to CM@easa.europa.eu.