The European Organisation for Civil Aviation Equipment
L'Organisation Européenne pour l'Equipement de l'Aviation Civile

# SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION

*Edition 2: This version includes amendment No 1 of October 19th, 1999.*



**ED-12B**

December 1992

EUROCAE
17 rue Hamelin
75783 PARIS Cedex 16
FRANCE

The European Organisation for Civil Aviation Equipment
L'Organisation Européenne pour l'Equipement de l'Aviation Civile

# CONSIDERATIONS SUR LE LOGICIEL EN VUE DE LA CERTIFICATION DES SYSTEMES ET EQUIPEMENTS DE BORD

*Edition 2 : Cette version inclut l' amendement N° 1 du 19 octobre, 1999.*



**ED-12B**

Décembre 1992

EUROCAE

17 rue Hamelin

75783 PARIS Cedex 16

FRANCE

# SOFTWARE CONSIDERATIONS IN AIRBORNE

# SYSTEMS AND EQUIPMENT CERTIFICATION

| | |
|---|---|
| *Edition 2:* | *This version includes amendment No 1 of October 19th, 1999.* |



**ED-12B**

December 1992

# CONSIDERATIONS SUR LE LOGICIEL EN VUE DE LA CERTIFICATION DES SYSTEMES ET EQUIPEMENTS DE BORD

*Edition 2 : Cette version inclut l' amendement N° 1 du 19 octobre, 1999.*



**ED-12B**

Décembre 1992

TABLE OF CONTENTS


SECTION 1 INTRODUCTION ..... 1

1.1 Purpose ..... 1

1.2 Scope ..... 1

1.3 Relationship to Other Documents ..... 1

1.4 How to Use This Document ..... 2

1.5 Document Overview ..... 3

Fugure 1-1 Document Overview ..... 3

SECTION 2 SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT ..... 4

2.1 Information Flow Between System and Software Life Cycle Processes ..... 4

2.1.1 Information Flow from System Processes to Software Processes ..... 4

2.1.2 Information Flow from Software Processes to System Processes ..... 6

2.2 Failure Condition and Software Level ..... 6

2.2.1 Failure Condition Categorization ..... 6

2.2.2 Software Level Definitions ..... 7

2.2.3 Software Level Determination ..... 7

2.3 System Architectural Considerations ..... 8

2.3.1 Partitioning ..... 8

2.3.2 Multiple-Version Dissimilar Software ..... 9

2.3.3 Safety Monitoring ..... 9

2.4 System Considerations for User-Modifiable Software, Option-Selectable Software and Commercial Off-The-Shelf Software ..... 10

2.5 System Design Considerations for Field-Loadable Software ..... 10

2.6 System Requirements Considerations for Software Verification ..... 11

2.7 Software Considerations in System Verification ..... 12

Figure 2-1 System Safety-Related Information Flow Between System and Software Life Cycle Processes ..... 5

SECTION 3 SOFTWARE LIFE CYCLE ..... 13

3.1 Software Life Cycle Processes ..... 13

3.2 Software Life Cycle Definition ..... 13

3.3 Transition Criteria Between Processes ..... 15

Figure 3-1 Example of Software Project Using four Different Development Sequences ..... 14

SECTION 4 SOFTWARE PLANNING PROCESS ..... 16

4.1 Software Planning Process Objectives ..... 16

TABLE DES MATIERES

SECTION 1 INTRODUCTION ..... 1
1.1 But ..... 1
1.2 Domaine d'application ..... 1
1.3 Relation avec d'Autres Documents ..... 1
1.4 Comment Utiliser Ce Document ..... 2
1.5 Vue Générale du Document ..... 3
Figure 1-1 Vue Générale du Document ..... 3

SECTION 2 ASPECTS SYSTEME LIES AU DEVELOPPEMENT DU LOGICIEL ..... 4
2.1 Flux d'Information Entre les Cycles de Vie du Système et du Logiciel ..... 4
2.1.1 Flux d'information des processus système vers les processus logiciel ..... 4
2.1.2 Flux d'information des processus logiciel vers les processus système ..... 6
2.2 Conditions de Panne et Niveau Logiciel ..... 6
2.2.1 Catégories des conditions de panne ..... 6
2.2.2 Définition des niveaux logiciels ..... 7
2.2.3 Détermination du niveau logiciel ..... 7
2.3 Considérations sur l'Architecture des Systèmes ..... 8
2.3.1 Partitionnement ..... 8
2.3.2 Logiciels à versions multiples dissimilaires ..... 9
2.3.3 Surveillance ..... 9
2.4 Considérations Relatives au Système pour les Logiciels Modifiables par l'Utilisateur, les Logiciels A option, et les Logiciels du commerce sur étagère ..... 10
2.5 Considération de Conception des Systèmes pour les Logiciels Téléchargeables ..... 10
2.6 Considérations de Système pour la Vérification des Logiciels ..... 11
2.7 Considérations Relatives au Logiciel dans la Vérification des Systèmes ..... 12
Figure 2-1 Flux d'Informations liées à la Sécurité entre le Processus de Cycle de Vie du Système et du Logiciel ..... 5

SECTION 3 CYCLE DE VIE DU LOGICIEL ..... 13
3.1 Processus du Cycle de Vie du Logiciel ..... 13
3.2 Définition du Cycle de Vie d'un Logiciel ..... 13
3.3 Critères de Transition entre Processus ..... 15
Figure 3-1 Exemple d'un Projet de Logiciel Utilisant quatre Séquence Différents de Développement 14

SECTION 4 PLANIFICATION DU LOGICIEL ..... 116
4.1 Objectifs de la Planification du Logiciel ..... 116

4.2 Software Planning Process Activities ..... 16
4.3 Software Plans ..... 17
4.4 Software Life Cycle Environment Planning ..... 18
4.4.1 Software Development Environment ..... 18
4.4.2 Language and Compiler Considerations ..... 19
4.4.3 Software Test Environment ..... 19
4.5 Software Development Standards ..... 19
4.6 Review and Assurance of the Software Planning Process ..... 20

SECTION 5 SOFTWARE DEVELOPMENT PROCESSES ..... 21
5.1 Software Requirements Process ..... 21
5.1.1 Software Requirements Process Objectives ..... 21
5.1.2 Software Requirements Process Activities ..... 22
5.2 Software Design Process ..... 22
5.2.1 Software Design Process Objectives ..... 22
5.2.2 Software Design Process Activities ..... 23
5.2.3 Designing for User-Modifiable Software ..... 23
5.3 Software Coding Process ..... 24
5.3.1 Software Coding Process Objectives ..... 24
5.3.2 Software Coding Process Activities ..... 24
5.4 Integration Process ..... 24
5.4.1 Integration Process Objectives ..... 24
5.4.2 Integration Process Activities ..... 25
5.4.3 Integration Considerations ..... 25
5.5 Traceability ..... 26

SECTION 6 SOFTWARE VERIFICATION PROCESS ..... 27
6.1 Software Verification Process Objectives ..... 27
6.2 Software Verification Process Activities ..... 28
6.3 Software Reviews and Analyses ..... 28
6.3.1 Reviews and Analyses of the High-Level Requirements ..... 29
6.3.2 Reviews and Analyses of the Low-Level Requirements ..... 29
6.3.3 Reviews and Analyses of the Software Architecture ..... 30
6.3.4 Reviews and Analyses of the Source Code ..... 30
6.3.5 Reviews and Analyses of the Outputs of the Integration Process ..... 31
6.3.6 Reviews and Analyses of the Test Cases, Procedures and Results ..... 31
6.4 Software Testing ..... 31
6.4.1 Test Environment ..... 33

4.2 Activités de Planification du Logiciel ..... 116
4.3 Les Plans ..... 17
4.4 Planification de l'Environnement du Cycle de Vie d'un Logiciel ..... 18
4.4.1 Environnement de développement du logiciel ..... 18
4.4.2 Considérations relatives aux langages et aux compilateurs ..... 19
4.4.3 Environnement de test du logiciel ..... 19
4.5 Règles de Développement des Logiciels ..... 19
4.6 Revues et Assurance de la Planification du Logiciel ..... 20

SECTION 5 DEVELOPPEMENT DU LOGICIEL ..... 21
5.1 Spécifications du Logiciel ..... 21
5.1.1 Objectifs des spécifications du logiciel ..... 21
5.1.2 Activités de définition des spécifications du logiciel ..... 22
5.2 Conception du Logiciel ..... 22
5.2.1 Objectifs de la conception du logiciel ..... 22
5.2.2 Activités de conception du logiciel ..... 23
5.2.3 Conception d'un logiciel modifiable par l'utilisateur ..... 23
5.3 Codage du Logiciel ..... 24
5.3.1 Objectifs du codage d'un logiciel ..... 24
5.3.2 Activités du codage d'un logiciel ..... 24
5.4 Intégration ..... 24
5.4.1 Objectifs de l'intégration ..... 24
5.4.2 Activités de l'intégration ..... 25
5.4.3 Considérations relatives à l'intégration ..... 25
5.5 Traçabilité ..... 26

SECTION 6 VERIFICATION DU LOGICIEL ..... 27
6.1 Objectifs du Processus de Vérification du Logiciel ..... 27
6.2 Activités du Processus de Vérification du Logiciel ..... 28
6.3 Revues et Analyses du Logiciel ..... 28
6.3.1 Revues et analyses des exigences de haut niveau ..... 29
6.3.2 Revues et analyses d'exigences de bas niveau ..... 29
6.3.3 Revues et analyses de l'architecture du logiciel ..... 30
6.3.4 Revues et analyses du Code Source ..... 30
6.3.5 Revues et analyses des sorties du processus d'intégration ..... 31
6.3.6 Revues et analyses des jeux, procédures, et résultats de test ..... 31
6.4 Tests du Logiciel ..... 31
6.4.1 Environnement de test ..... 33

6.4.2 Requirements-Based Test Case Selection ....33
6.4.3 Requirements-Based Testing Methods ....34
6.4.4 Test Coverage Analysis ....35
Figure 6-1 Software Testing Process ....32

SECTION 7 SOFTWARE CONFIGURATION MANAGEMENT PROCESS ....37
7.1 Software Configuration Management Process Objectives ....37
7.2 Software Configuration Management Process Activities ....37
7.2.1 Configuration Identification ....38
7.2.2 Baselines and Traceability ....38
7.2.3 Problem Reporting, Tracking and Corrective Action ....39
7.2.4 Change Control ....39
7.2.5 Change Review ....40
7.2.6 Configuration Status Accounting ....40
7.2.7 Archive, Retrieval and Release ....40
7.2.8 Software Load Control ....41
7.2.9 Software Life Cycle Environment Control ....41
7.3 Data Control Categories ....42
Table 7-1 SCM Process Objectives Associated with CC1 and CC2 Data ....42

SECTION 8 SOFTWARE QUALITY ASSURANCE PROCESS ....43
8.1 Software Quality Assurance Process Objectives ....43
8.2 Software Quality Assurance Process Activities ....43
8.3 Software Conformity Review ....44

SECTION 9 CERTIFICATION LIAISON PROCESS ....45
9.1 Means of Compliance and Planning ....45
9.2 Compliance Substantiation ....45
9.3 Minimum Software Life Cycle Data That Is Submitted tO Certification Authority ....45
9.4 Software Life Cycle Data Related to Type Design ....46

SECTION 10 OVERVIEW OF AIRCRAFT AND ENGINE CERTIFICATION ....47
10.1 Certification Basis ....47
10.2 Software Aspects of Certification ....47
10.3 Compliance Determination ....47

SECTION 11 SOFTWARE LIFE CYCLE DATA ....48
11.1 Plan for Software Aspects of Certification ....49

6.4.2 Choix de jeux de test fondés sur les exigences ..... 33
6.4.3 Méthodes de test fondées sur les exigences ..... 34
6.4.4 Analyse de la couverture de test ..... 35
Figure 6-1 Test du Logiciel ..... 32

SECTION 7 GESTION DE CONFIGURATION DU LOGICIEL ..... 37
7.1 Objectifs du Processus de Gestion de Configuration du Logiciel ..... 37
7.2 Activités du Processus de Gestion de Configuration du Logiciel ..... 37
7.2.1 Identification de configuration ..... 38
7.2.2 Référentiels et traçabilité ..... 38
7.2.3 Compte rendu des anomalies, suivi et actions correctives ..... 39
7.2.4 Gestion des modifications ..... 39
7.2.5 Revue des modifications ..... 40
7.2.6 Suivi des états de configuration ..... 40
7.2.7 Archivage, restauration et mise à disposition ..... 40
7.2.8 Contrôle du chargement du logiciel ..... 41
7.2.9 Contrôle de l'environnement de développement du logiciel ..... 41
7.3 Catégories de Contrôle deS Données ..... 42
Figure 7-1 Objectifs du Processus de SCM Associés aux Données CC1 et CC2 ..... 42

SECTION 8 ASSURANCE QUALITE DU LOGICIEL ..... 43
8.1 Objectifs du Processus d'Assurance Qualité du Logiciel ..... 43
8.2 Activités du Processus d'Assurance Qualité du Logiciel ..... 43
8.3 Revue de Conformité du Logiciel ..... 44

SECTION 9 COORDINATION POUR LA CERTIFICATION ..... 45
9.1 Moyens de Conformité et Planification ..... 45
9.2 Justification de Conformité ..... 45
9.3 Données Minimales du Cycle de Vie du Logiciel Soumises à l'Autorité de Certification ..... 45
9.4 Données du Cycle de Vie du Logiciel Liées à une definition de Type ..... 46

SECTION 10 VUE GENERALE SUR LA CERTIFICATION DES AERONEFS ET DES MOTEURS ..... 47
10.1 Bases de Certification ..... 47
10.2 Aspects Logiciels de la Certification ..... 47
10.3 Etablissement de la Conformité ..... 47

SECTION 11 DONNEES DU CYCLE DE VIE DU LOGICIEL ..... 48
11.1 Plan des aspects logiciels de la Certification ..... 49

11.2 Software Development Plan ..... 49
11.3 Software Verification Plan ..... 50
11.4 Software Configuration Management Plan ..... 51
11.5 Software Quality Assurance Plan ..... 52
11.6 Software Requirements Standards ..... 52
11.7 Software Design Standards ..... 53
11.8 Software Code Standards ..... 53
11.9 Software Requirements Data ..... 53
11.10 Design Description ..... 54
11.11 Source Code ..... 54
11.12 Executable Object Code ..... 54
11.13 Software Verification Cases and Procedures ..... 55
11.14 Software Verification Results ..... 55
11.15 Software Life Cycle Environment Configuration Index ..... 55
11.16 Software Configuration Index ..... 55
11.17 Problem Reports ..... 56
11.18 Software Configuration Management Records ..... 56
11.19 Software Quality Assurance Records ..... 56
11.20 Software Accomplishment Summary ..... 57

SECTION 12 ADDITIONAL CONSIDERATIONS ..... 58
12.1 Use of Previously Developed Software ..... 58
12.1.1 Modifications to Previously Developed Software ..... 58
12.1.2 Change of Aircraft Installation ..... 58
12.1.3 Change of Application or Development Environment ..... 59
12.1.4 Upgrading A Development Baseline ..... 59
12.1.5 Software Configuration Management Considerations ..... 60
12.1.6 Software Quality Assurance Considerations ..... 60
12.2 Tool Qualification ..... 60
12.2.1 Qualification Criteria for Software Development Tools ..... 61
12.2.2 Qualification Criteria for Software Verification Tools ..... 62
12.2.3 Tool Qualification Data ..... 62
12.2.4 Tool Qualification Agreement ..... 63
12.3 Alternative Methods ..... 63
12.3.1 Formal Methods ..... 64
12.3.2 Exhaustive Input Testing ..... 65
12.3.3 Considerations for Multiple-Version Dissimilar Software Verification ..... 65
12.3.4 Software Reliability Models ..... 67
12.3.5 Product Service History ..... 67

11.2 Plan de Développement du Logiciel ..... 49
11.3 Plan de Vérification du Logiciel ..... 50
11.4 Plan de Gestion de Configuration du Logiciel ..... 51
11.5 Plan d'Assurance Qualité du Logiciel ..... 52
11.6 Règles de Spécification du Logiciel ..... 52
11.7 Règles de Conception du Logiciel ..... 53
11.8 Règles de Codage du Logiciel ..... 53
11.9 Données de spécification du Logiciel ..... 53
11.10 Description de Conception ..... 54
11.11 Code Source ..... 54
11.12 Code Objet Exécutable ..... 54
11.13 Jeux et Procédures de Vérification du Logiciel ..... 55
11.14 Résultats de Vérification du Logiciel ..... 55
11.15 Répertoire de la configuration de l'Environnement du Logiciel ..... 55
11.16 Répertoire de la configuration du Logiciel ..... 55
11.17 Rapports d'anomalies ..... 56
11.18 Documents de Gestion de Configuration du Logiciel ..... 56
11.19 Documents d'Assurance Qualité du Logiciel ..... 56
11.20 Résumé des travaux réalisés ..... 57

SECTION 12 CONSIDERATIONS COMPLEMENTAIRES ..... 58
12.1 Utilisation de Logiciel Développé Antérieurement ..... 58
12.1.1 Modifications d'un logiciel développé antérieurement ..... 58
12.1.2 Modification de l'installation d'un aéronef ..... 58
12.1.3 Modification de l'environnement de développement ou de son utilisation ..... 59
12.1.4 Mise à niveau d'un référentiel ..... 59
12.1.5 Considérations relatives à la gestion de configuration ..... 600
12.1.6 Considérations relatives à l'assurance qualité ..... 600
12.2 Qualification d'Outil ..... 600
12.2.1 Critères de qualification d'outils de développement ..... 611
12.2.2 Critères de qualification des outils de vérification ..... 622
12.2.3 Données de qualification d'outil ..... 622
12.2.4 Agrément de qualification d'outil ..... 633
12.3 Méthodes de substitution ..... 633
12.3.1 Méthodes formelles ..... 644
12.3.2 Test exhaustif des données d'entrée ..... 655
12.3.3 Considérations relatives aux logiciels à versions multiples dissimilaires ..... 655
12.3.4 Modèles de fiabilité de logiciel ..... 677
12.3.5 Historique du produit en service ..... 677

ANNEX A PROCESS OBJECTIVES AND OUTPUTS BY SOFTWARE LEVEL ..... 69
Table A-1 Software Planning Process. ..... 70
Table A-2 Software Development Processes. ..... 71
Table A-3 Verification of Outputs of Software Requirements Process. ..... 72
Table A-4 Verification of outputs of Software Design Process. ..... 73
Table A-5 Verification of Outputs of Software Coding & Integration Processes. ..... 74
Table A-6 Testing of Outputs of Integration Process. ..... 75
Table A-7 Verification of Verification Process Results. ..... 76
Table A-8 Software Configuration Management Process. ..... 77
Table A-9 Software Quality Assurance Process. ..... 78
Table A-10 Certification Liaison Process. ..... 79

ANNEX B ACRONYMS AND GLOSSARY OF TERMS ..... 80

APPENDIX A BACKGROUND OF DOCUMENT ED-12 ..... 89
1.0 Prior Document Version History ..... 89
2.0 RTCA / EUROCAE Committee Activities in the Production of This Document ..... 89
3.0 Summary Of Differences between ED-12B and ED-12A ..... 91

APPENDIX B COMMITTEE MEMBERSHIP ..... 92

APPENDIX C IMPROVEMENT SUGGESTION FORM ..... 96

AMENDMENT No 1 ..... 97

ANNEXE A OBJECTIFS ET PRODUITS DES PROCESSUS PAR NIVEAU LOGICIEL ..... 699
Tableau A-1 Planification du logiciel ..... 70
Tableau A-2 Développement du Logiciel ..... 71
Tableau A-3 Vérification des Produits des Spécifications du Logiciel ..... 72
Tableau A-4 Vérification des Produits de la Conception Logiciel ..... 73
Tableau A-5 Vérification des Produits du Codage et de l'Intégration du Logiciel ..... 74
Tableau A-6 Test des Produits de l'Intégration ..... 75
Tableau A-7 Vérification des Produits de la Vérification ..... 76
Tableau A-8 Gestion de Configuration ..... 77
Tableau A-9 Assurance Qualité ..... 78
Tableau A-10 Coordination pour la Certification ..... 79

ANNEXE B ACRONYMES ET GLOSSAIRE ..... 80

APPENDICE A ORIGINES DU DOCUMENT EUROCAE ED-12B ..... 89
1.0 Historique des Versions Anterieures du Document ..... 89
2.0 Activités du Comité RTCA/EUROCAE dans l'élaboration du présent document ..... 89
3.0 Résumé des différences entre les documents ED-12B et ED-12A ..... 91

APPENDICE B COMITE MEMBRE ..... 92

APPENDICE C FORMULAIRE DE SUGGESTION D'AMELIORATION ..... 96

AMENDEMENT N° 1 ..... 97

# FOREWORD

1. This document, jointly prepared by EUROCAE Working Group 12 and RTCA Special Committee 167 was accepted by the Council of EUROCAE on December 10, 1992. EUROCAE ED-12B is identical to RTCA DO-178B.

2. EUROCAE is an international non-profit making organisation. Membership is open to European users and manufacturers of equipment for aeronautics, trade associations, national civil aviation administrations and, under certain conditions, non-European members. Its work programme is directed to the study of technical problems facing users and manufacturers of equipment for aeronautics and all related topics. It aims to contribute at national and international levels to the solution of such problems.

3. The findings of EUROCAE are resolved after discussion among members of EUROCAE, European airlines and interested administrations, and in collaboration with the RTCA (Requirements and Technical Concepts for Aviation), Washington, DC, and/or the SAE (Society of Automotive Engineers), Warrendale, PA, USA, through their appropriate committees.

4. EUROCAE Minimum operational performance specifications are recommendations only. EUROCAE is not an official body of the European Governments; its recommendations are valid as statements of official policy only when adopted by a particular government or conference of governments.

5. Copies of this document may be obtained from:

EUROCAE
17 rue Hamelin
75783 PARIS CEDEX 16
France

Phone: 33 1 45 05 78 11
Fax: 33 1 45 05 72 30
E-mail: eurocae@eurocae.com
Website: www.eurocae.org

# PREFACE

1. Le présent document préparé en commun par le Groupe de Travail 12 de l'EUROCAE et le Comité Spécial 167 de la RTCA a été approuvé par le Comité de Direction le 10 décembre 1992. EUROCAE ED-12B est identique au RTCA DO-178B.

2. L'EUROCAE est une organisation internationale à but non lucratif. Peuvent en être membres les utilisateurs et les fabricants en Europe d'équipements destinés à l'aéronautique, les associations professionnelles, les administrations d'aviation civile, et, sous certaines conditions, des membres non européens. Son programme de travail est dirigé vers l'étude des problèmes techniques qui se posent aux utilisateurs et aux fabricants d'équipements destinés à l'aéronautique et de toute autre question s'y rapportant. Elle essaie de contribuer à la solution de tels problèmes aux niveaux nationaux et international.

3. Les décisions de l'EUROCAE sont prises après discussion entre les membres de l'EUROCAE, les compagnies aériennes européennes et les administrations intéressées, en collaboration avec la RTCA (Requirements and Technical Concepts for Aviation), Washington D.C., et/ou la SAE (Society of Automotive Engineers), Warrendale, PA, USA, par l'intermédiaire de leurs comités appropriés.

4. Les spécifications de performances opérationnelles minimales de l'EUROCAE ne constituent que des recommandations : l'EUROCAE n'est pas un organisme officiel des gouvernements européens, ses recommandations par conséquent, ne sont validées en tant que déclaration de politique officielle que lorsqu'elles sont adoptées par un gouvernement particulier ou par une conférence de gouvernements.

5. Des exemplaires de ce document peuvent être obtenus sur demande à :

EUROCAE
17, rue Hamelin
75783 PARIS CEDEX 16
France

Phone: 33 1 45 05 78 11
Fax: 33 1 45 05 72 30
E-mail: eurocae@eurocae.com
Website: www.eurocae.org

# SECTION 1

# INTRODUCTION

The rapid increase in the use of software in airborne systems and equipment used on aircraft and engines in the early 1980s resulted in a need for industry-accepted guidance for satisfying airworthiness requirements. ED-12, "Software Considerations in Airborne Systems and Equipment Certification," was written to satisfy this need.

This document, now revised in the light of experience, provides the aviation community with guidance for determining, in a consistent manner and with an acceptable level of confidence, that the software aspects of airborne systems and equipment comply with airworthiness requirements. As software use increases, technology evolves and experience is gained in the application of this document, this document will be reviewed and revised. Appendix A contains a history of this document.

## 1.1 PURPOSE

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

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

## 1.2 SCOPE

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

Since certification issues are discussed only in relation to the software life cycle, the operational aspects of the resulting software are not discussed. For example, the certification aspects of user-modifiable data are beyond the scope of this document.

This document does not provide guidelines concerning the structure of the applicant's organization, the relationships between the applicant and its suppliers, or how the responsibilities are divided. Personnel qualification criteria are also beyond the scope of this document.

## 1.3 RELATIONSHIP TO OTHER DOCUMENTS

In addition to the airworthiness requirements, various national and international standards for software are available. In some communities, compliance with these standards may be required. However, it is outside the scope of this document to invoke specific national or international standards, or to propose a means by which these standards might be used as an alternative or supplement to this document.

# SECTION 1

# INTRODUCTION

L'accroissement rapide de l'utilisation des logiciels dans les systèmes et équipements de bord utilisés sur les aéronefs et les moteurs s'est traduit au début des années 1980 par un besoin de directives acceptées par l'Industrie pour satisfaire les exigences de navigabilité. Le document ED-12 "Etude et Homologation du Logiciel des Systèmes et Equipements de Bord" avait été rédigé pour répondre à ce besoin.

Ce document, maintenant révisé à la lumière de l'expérience, fournit à la Communauté Aéronautique des directives pour déterminer, de manière cohérente et avec un degré acceptable de confiance, si les aspects relatifs aux logiciels des systèmes et équipements de bord satisfont les exigences de navigabilité. Ce document sera revu et révisé au fur et à mesure de la généralisation de l'utilisation des logiciels, de l'évolution de la technologie et en tenant compte de l'expérience acquise. L'Appendice A retrace l'historique de ce document.

## 1.1 BUT

Ce document a pour but de fournir des recommandations pour la réalisation de logiciels destinés aux systèmes et équipements de bord, logiciels qui doivent remplir une fonction définie avec, en ce qui concerne la sécurité, un degré de confiance en accord avec les exigences de navigabilité. Ces recommandations se présentent sous la forme :

- D'objectifs pour les processus du cycle de vie du logiciel.
- De descriptions des activités et des considérations de conception pour satisfaire ces objectifs.
- De descriptions des preuves qui indiquent que les objectifs ont été satisfaits.

## 1.2 DOMAINE D'APPLICATION

Ce document traite des aspects relatifs à la certification de navigabilité, qui se rapportent à la réalisation des logiciels des systèmes et équipements de bord utilisés sur les aéronefs ou les moteurs. Dans l'exposé de ces aspects, le cycle de vie du système et sa relation avec le cycle de vie du logiciel sont décrits de manière à aider à la compréhension du processus de certification. Il n'est prévu une description complète ni du cycle de vie des systèmes, comprenant les processus d'analyse de sécurité des systèmes et de validation, ni des processus de certification des aéronefs et moteurs.

Etant donné que les questions de certification ne sont traitées qu'en relation avec le cycle de vie du logiciel, les aspects relatifs à l'utilisation opérationnelle du logiciel qui en résulte ne seront pas traités. Par exemple, les aspects relatifs à la certification se rapportant aux données modifiables par le postulant ne sont pas couverts par ce document.

Ce document ne donne pas de recommandations relatives à l'organisation du postulant, aux rapports entre le postulant et ses fournisseurs, ou à la manière de répartir les responsabilités. Les critères de qualification du personnel ne sont pas couverts par ce document.

## 1.3 RELATION AVEC D'AUTRES DOCUMENTS

Outre les exigences de navigabilité, il existe diverses normes nationales et internationales relatives aux logiciels. Dans certains cas, le respect de ces normes peut être imposé. Toutefois, il n'appartient pas à ce document de faire référence à des normes nationales ou internationales spécifiques, ni de proposer un mode d'utilisation de ces normes en remplacement ou en supplément de ce document.

Where this document uses the term "standards," it should be interpreted to mean the use of project-specific standards as applied by the airborne system, airborne equipment, engine, or aircraft manufacturer. Such standards may be derived from general standards produced or adopted by the manufacturer for its activities.

## 1.4 HOW TO USE THIS DOCUMENT

These points need to be noted when using this document:

- Explanatory text is included to aid the reader in understanding the topic under discussion. For example, section 2 provides information necessary to understand the interaction between the system life cycle and software life cycle. Similarly, section 3 is a description of the software life cycle and section 10 is an overview of aircraft and engine certification.
- This document is intended to be used by the international aviation community. To aid such use, references to specific national regulations and procedures are minimized. Instead, generic terms are used. For example, the term "certification authority" is used to mean the organization or person granting approval on behalf of the country responsible for aircraft or engine certification. Where a second country or a group of countries validates or participates in this certification, this document may be used with due recognition given to bilateral agreements or memoranda of understanding between the countries involved.
- This document recognizes that the guidelines herein are not mandated by law, but represent a consensus of the aviation community. It also recognizes that alternative methods to the methods described herein may be available to the applicant. For these reasons, the use of words such as "shall" and "must" is avoided.
- This document states the objectives for the software levels, as defined in paragraph 2.2.2. Annex A specifies the variation in these objectives by software level. If an applicant adopts this document for certification purposes, it may be used as a set of guidelines to achieve these objectives.
- Section 11 contains the data generally produced to aid the software aspects of the certification process. The names of the data are denoted in the text by capitalization of the first letter of each word in the name. For example, Source Code.
- Section 12 discusses additional considerations including guidance for the use of previously developed software, for tool qualification, and for the use of alternative methods to those described in sections 2 through 11. Section 12 may not apply to every certification.
- The tables for software level variation and the glossary are contained in Annexes, and are normative parts of this document. Other material is contained in Appendices, and are informative parts of this document.
- In cases where examples are used to indicate how the guidelines might be applied, either graphically or through narrative, the examples are not to be interpreted as the preferred method.
- A list of items does not imply the list is all-inclusive.
- Notes are used in this document to provide explanatory material, emphasize a point, or draw attention to related items which are not entirely within context. Notes do not contain guidance.

Quand ce document utilise le terme de "règles", il faut l'interpréter comme signifiant l'utilisation de règles spécifiques à un projet et utilisées par le constructeur du système de bord, de l'équipement de bord, du moteur, ou de l'aéronef. De telles règles peuvent être établies à partir de normes générales produites ou adoptées par le constructeur pour ses activités.

*NOTA : Le terme anglais "standards" a été systématiquement traduit par "règles".*

## 1.4 COMMENT UTILISER CE DOCUMENT

Les points suivants sont à noter pour utiliser ce document :

- Un texte explicatif est inclus afin d'aider le lecteur à comprendre le sujet traité. Par exemple, la section 2 fournit les informations nécessaires à la compréhension de l'interaction entre le cycle de vie du système et le cycle de vie du logiciel. De même, la section 3 constitue une description du cycle de vie du logiciel et la section 10 est une vue générale sur la certification des aéronefs et des moteurs.
- Ce document est destiné à être utilisé par la communauté aéronautique internationale. Afin d'aider à une telle utilisation, les références à des réglementations et procédures nationales spécifiques ont été réduites au minimum. Au contraire, on a employé des termes génériques. Par exemple, le terme "Autorité de certification" est utilisé pour désigner l'organisation ou la personne délivrant l'attestation au nom du pays responsable de la certification de l'aéronef ou du moteur. Dans le cas où un second pays ou un groupe de pays valident cette certification ou y participent, il est possible d'utiliser ce document en reconnaissant dûment les accords ou protocoles entre les pays concernés.
- Ce document reconnaît que les présentes recommandations n'ont pas force de loi, mais représentent un consensus de la Communauté Aéronautique. Il reconnaît aussi que le postulant peut disposer d'autres méthodes que celles décrites ici. Pour ces raisons, on a évité l'utilisation de mots tels que "devra (shall, must)" ou de tournures à caractère impératif.

  *NOTA : Pour éviter l'usage systématique de locutions telles que "il est recommandé que (de)" ou "doit de préférence", le verbe anglais should a été traduit systématiquement par "doit".*

- Ce document précise les objectifs par niveau logiciel, selon les définitions du paragraphe 2.2.2. L'Annexe A spécifie les variations entre ces objectifs par niveau logiciel. Si un postulant adopte ce document dans un but de certification, il peut le considérer comme un ensemble de recommandations pour atteindre ces objectifs.
- La section 11 décrit les données généralement fournies pour aider à la présentation des aspects relatifs au logiciel dans le processus de certification. Les noms de ces données sont mis en exergue dans le texte par emploi d'une majuscule pour la première lettre de chaque mot. Par exemple, Code Source.
- La section 12 traite de considérations complémentaires comprenant des directives pour l'utilisation de logiciel développé antérieurement, pour la qualification d' outil, et pour l'utilisation de méthodes de substitution à celles décrites dans les sections 2 à 11. La section 12 peut ne pas s'appliquer à certaines certifications.
- Les tableaux de variations par niveau logiciel et le glossaire se trouvent dans les annexes A et B, et constituent des parties normatives de ce document. D'autres informations se trouvent dans les appendices, et constituent des parties informatives de ce document.
- Lorsque des exemples sont utilisés pour indiquer comment appliquer les recommandations, soit graphiquement, soit de manière narrative, ces exemples ne doivent pas être interprétés comme étant des méthodes préférentielles.
- Des notes sont utilisées dans ce document pour donner des explications, souligner un point, attirer l'attention sur des sujets associés ne se trouvant pas entièrement dans le contexte du paragraphe. Les notes ne contiennent pas de directives.

## 1.5 DOCUMENT OVERVIEW

Figure 1-1 is a pictorial overview of this document's sections and their relationship to each other.

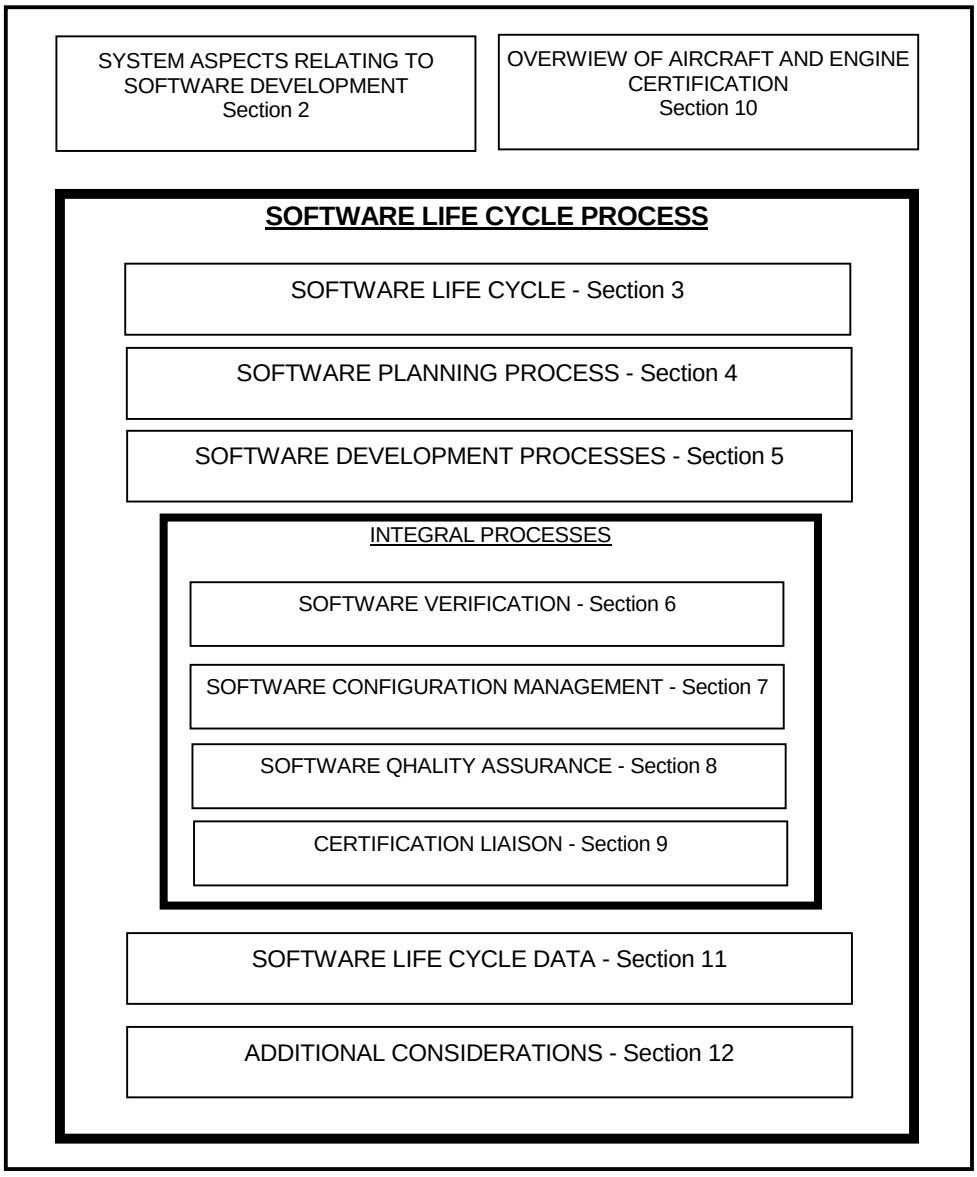



FIGURE 1-1: DOCUMENT OVERVIEW

## 1.5 VUE GENERALE DU DOCUMENT

La Figure 1-1 donne une vue générale des différentes sections de ce document et de leurs relations entre elles.

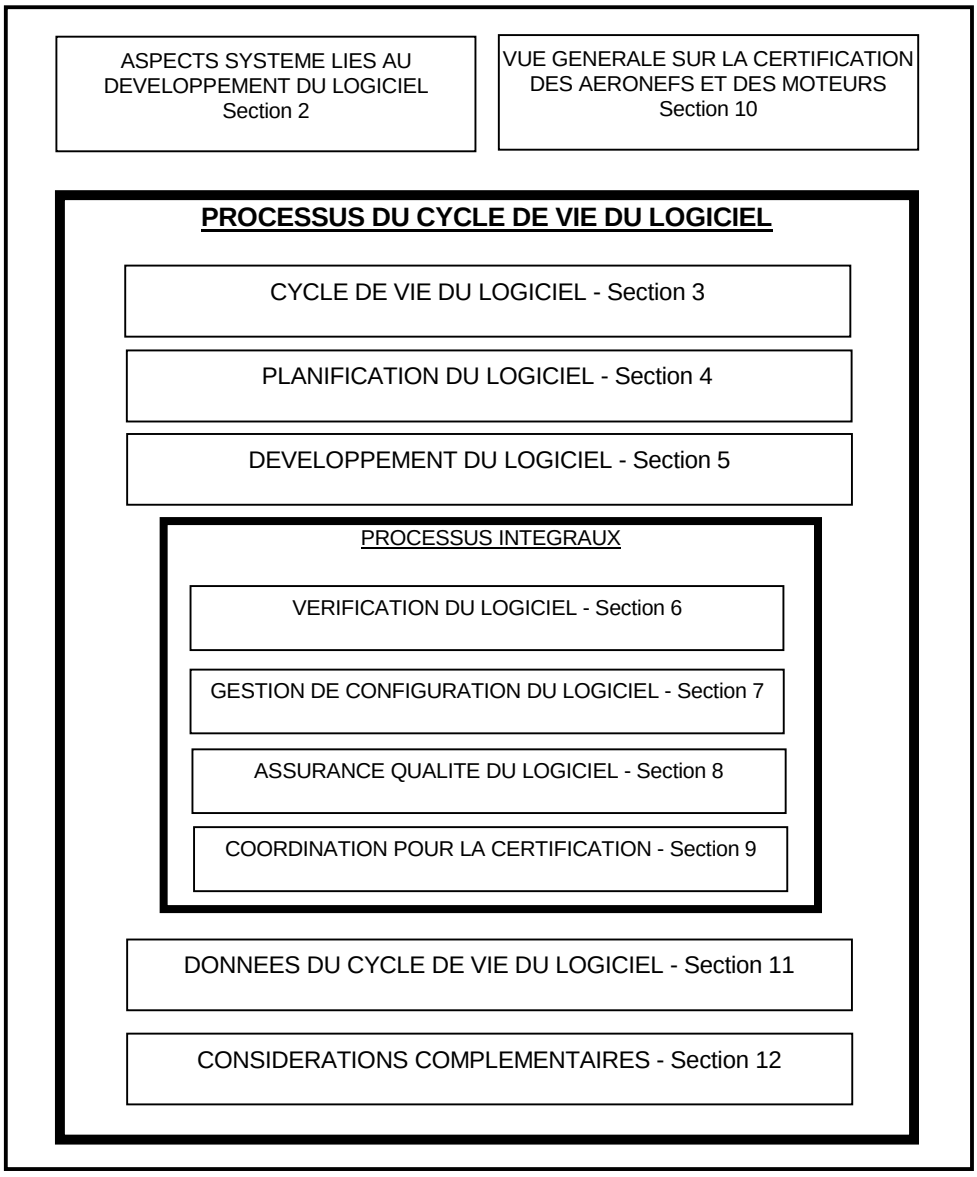



FIGURE 1-1 : VUE GENERALE DU DOCUMENT

# SECTION 2

# SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT

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

- Exchange of data between the system and software life cycle processes (subsection 2.1).
- Categorization of failure conditions, definition of software levels, and software level determination (subsection 2.2).
- System architectural considerations (subsection 2.3).
- System considerations for user-modifiable software, option-selectable software, and commercial off-the-shelf software (subsection 2.4).
- System design considerations for field-loadable software (subsection 2.5).
- System requirements considerations for software verification (subsection 2.6).
- Software considerations in system verification (subsection 2.7).

## 2.1 INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES

Figure 2-1 is an overview of the safety aspects of the information flow between system life cycle processes and the software life cycle processes. Due to interdependence of the system safety assessment process and the system design process, the flow of information described in these sections is iterative.

### 2.1.1 Information Flow from System Processes to Software Processes

The system safety assessment process determines and categorizes the failure conditions of the system. Within the system safety assessment process, an analysis of the system design defines safety related requirements that specify the desired immunity from, and system responses to, these failure conditions. These requirements are defined for hardware and software to preclude or limit the effects of faults, and may provide fault detection and fault tolerance. As decisions are being made during the hardware design process and software development processes, the system safety assessment process analyzes the resulting system design to verify that it satisfies the safety-related requirements.

# SECTION 2

## ASPECTS SYSTEME LIES AU DEVELOPPEMENT DU LOGICIEL

Cette section traite des aspects du cycle de vie du système nécessaires à la compréhension du cycle de vie du logiciel :

- Echange de données entre les cycles de vie du système et du logiciel (sous-section 2.1).
- Classement par catégorie des conditions de panne, définition des niveaux logiciels et règles de détermination du niveau logiciel (sous-section 2.2).
- Considérations sur l'architecture des systèmes (sous-section 2.3).
- Considérations sur les systèmes intégrant des logiciels modifiables par le postulant, des logiciels à option, et des logiciels du commerce sur étagère (sous-section 2.4).
- Considérations sur la conception des systèmes intégrant des logiciels téléchargeables (sous-section 2.5).
- Considérations sur les spécifications des systèmes relatives à la vérification du logiciel (sous-section 2.6).
- Considérations relatives aux logiciels dans la vérification des systèmes (sous-section 2.7).

### 2.1 FLUX D'INFORMATION ENTRE LES CYCLES DE VIE DU SYSTEME ET DU LOGICIEL

La Figure 2.1 constitue une vue générale des aspects relatifs à la sécurité du flux d'information entre les cycles de vie du système et du logiciel. En raison de l'interdépendance entre l'analyse de sécurité du système et la conception du système, les informations décrites dans ces sections sont produites de façon itérative.

#### 2.1.1 Flux d'information des processus système vers les processus logiciel

L'analyse de sécurité du système détermine et classe les conditions de panne du système. Dans le cadre de l'analyse de sécurité du système, une analyse de la conception du système définit les exigences liées à la sécurité, en spécifiant l'immunité désirée et les réponses du système aux conditions de panne. Ces exigences sont définies pour le matériel et le logiciel dans le but de prévenir ou de limiter les effets des défauts, et peuvent fournir une détection des défauts et une tolérance aux défauts. Comme les décisions sont prises au cours de la conception du matériel et du développement du logiciel, l'analyse de sécurité du système analyse la conception qui en résulte afin de vérifier si celle-ci répond aux exigences liées à la sécurité.

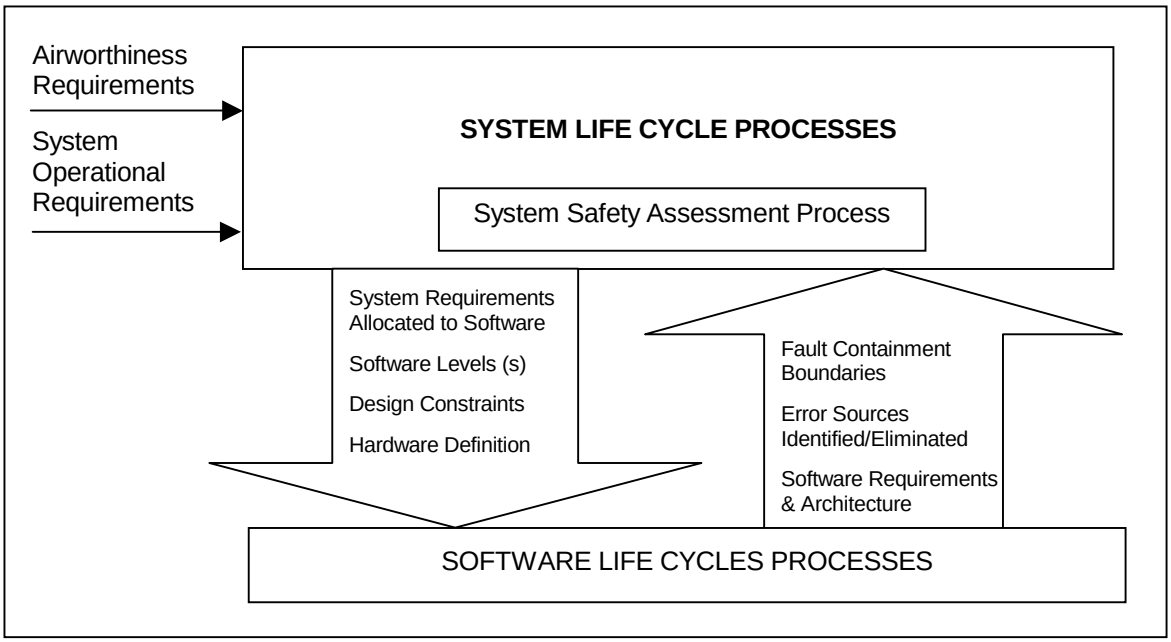



FIGURE 2-1: SYSTEM SAFETY-RELATED INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES

*NOTE: At the time of publication of this document, guidelines for the system life cycle processes were under development by an international committee. While every attempt was made to keep the inter-process information flows and definitions compatible, some differences may exist between the final published documents. Any differences will be reconciled in future revisions of the documents.*

The safety-related requirements are a part of the system requirements which are inputs to the software life cycle processes. To ensure that the safety-related requirements are properly implemented throughout the software life cycle, the system requirements typically include or reference:

- The system description and hardware definition.
- Certification requirements, including applicable Federal Aviation Regulations (FAR - United States), Joint Aviation Regulations (JAR - Europe), Advisory Circulars (United States), etc.
- System requirements allocated to software, including functional requirements, performance requirements, and safety-related requirements.
- Software level(s) and data substantiating their determination, failure conditions, their categories, and related functions allocated to software.
- Safety strategies and design constraints, including design methods, such as, partitioning, dissimilarity, redundancy or safety monitoring.
- If the system is a component of another system, the safety-related requirements and failure conditions for that system.

System life cycle processes may specify requirements for the software life cycle processes to aid system verification activities.

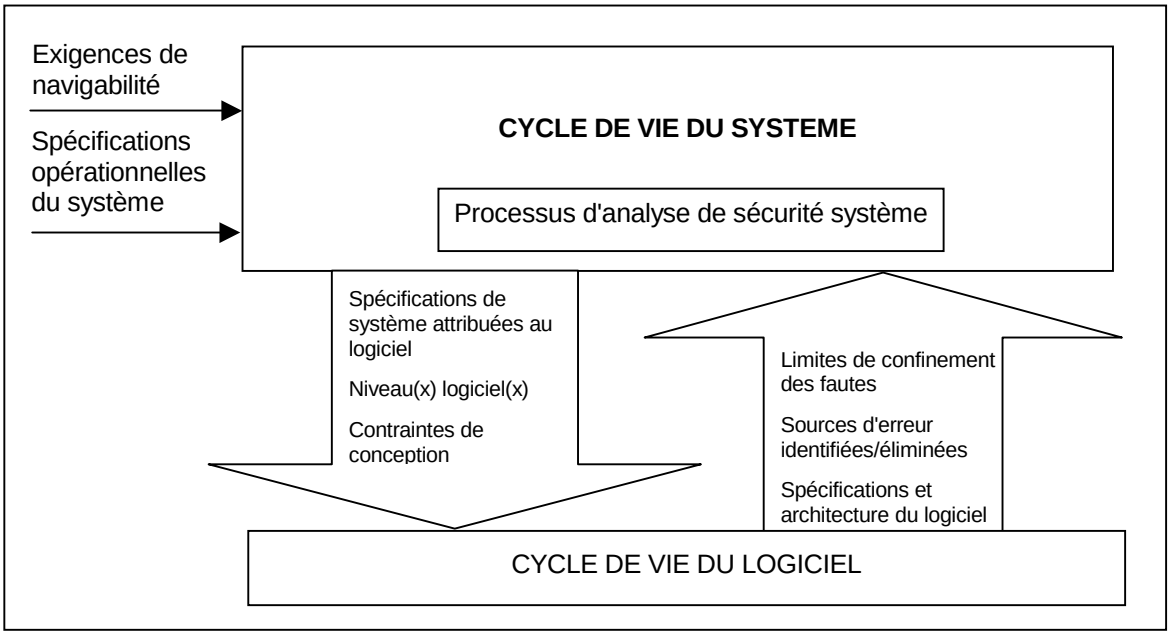



FIGURE 2-1 : FLUX D'INFORMATIONS LIEES A LA SECURITE ENTRE LES PROCESSUS DE CYCLE DE VIE DU SYSTEME ET DU LOGICIEL

*NOTA : A la date de publication de ce document, les recommandations pour le cycle de vie du système étaient en cours de développement au sein d'un Comité International. Bien que tout ait été fait pour maintenir la compatibilité des flux d'information entre les différents processus et les définitions, il peut exister certaines différences entre les documents définitifs publiés. Ces différences disparaîtront dans les révisions futures des documents.*

Les exigences liées à la sécurité constituent une partie des exigences du système et constituent des données d'entrée pour le cycle de vie du logiciel. Afin de s'assurer que les exigences relatives à la sécurité sont bien prises en compte tout au long du cycle de vie du logiciel, les spécifications du système comprennent habituellement ou font référence à :

- La description du système et la définition du matériel.
- Les exigences de certification, telles que les règlements applicables FAR ou JAR, les AC ou les AMJ, etc.
- Les spécifications du système allouées au logiciel, telles que les spécifications fonctionnelles, les spécifications de performances, et les exigences liées à la sécurité.
- Le(s) niveau(x) logiciel(s) et les éléments de justification, les conditions de panne, leur catégorie, et les fonctions allouées au logiciel.
- Les stratégies de prise en compte de la sécurité et les contraintes de conception, y compris les méthodes telles que partitionnement, dissimilarité, redondance, ou surveillance.
- Si le système est un composant d'un autre système, les exigences relatives à la sécurité et les conditions de panne de ce dernier.

Le cycle de vie du système peut également générer des exigences destinées à aider les activités de vérification du système.

### 2.1.2 Information Flow from Software Processes to System Processes

The system safety assessment process determines the impact of the software design and implementation on system safety using information provided by the software life cycle processes. This information includes fault containment boundaries, software requirements, software architecture, and error sources that may have been detected or eliminated through software architecture or by the use of tools or by other methods used in the software design process. Traceability between system requirements and software design data is important to the system safety assessment process.

Modifications to the software may affect system safety and, therefore, need to be identified to the system safety assessment process for evaluation.

## 2.2 FAILURE CONDITION AND SOFTWARE LEVEL

Guidance follows concerning system failure condition categories, the definition of software levels, the relationship between software levels and failure condition categories, and how software level is determined.

The failure condition category of a system is established by determining the severity of failure conditions on the aircraft and its occupants. An error in software may cause a fault that contributes to a failure condition. Thus, the level of software integrity necessary for safe operation is related to the system failure conditions.

### 2.2.1 Failure Condition Categorization

For a complete definition of failure condition categories, refer to the applicable regulations and guidance material, Federal Aviation Administration AC 25-1309-1A and/or the Joint Aviation Authorities AMJ 25-1309, as amended. The failure condition categories listed are derived from this guidance material and are included to assist in the use of this document. The categories are:

a. Catastrophic: Failure conditions which would prevent continued safe flight and landing.

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

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

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

   (3) adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants.

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

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

2.1.2 Flux d'information des processus logiciel vers les processus système

L'analyse de sécurité du système détermine l'impact de la conception et de la réalisation du logiciel sur la sécurité du système au moyen d'informations fournies par les processus du cycle de vie du logiciel. Ces informations comprennent les limites de confinement des défauts, les spécifications du logiciel, l'architecture du logiciel, et les sources d'erreurs qui auraient pu être détectées ou éliminées par des choix d'architecture, ou par l'usage d'outils, ou par d'autres méthodes lors de la conception du logiciel. La traçabilité entre les spécifications du système et les produits de la conception du logiciel est importante pour l'analyse de sécurité du système.

Les modifications apportées au logiciel peuvent affecter la sécurité du système et doivent donc être communiquées au processus d'analyse de sécurité du système.

## 2.2 CONDITIONS DE PANNE ET NIVEAU LOGICIEL

Les directives ci-dessous concernent les catégories de conditions de panne du système, la définition des niveaux logiciels, la relation entre niveaux logiciels et catégories de conditions de panne, et le mode de détermination des niveaux logiciels.

Les catégories sont établies en déterminant la sévérité des conditions de panne d'un système sur l'aéronef et ses occupants. Une erreur dans la spécification, la conception et le codage du logiciel peut être la cause d'un défaut contribuant à une condition de panne. Le niveau d'intégrité du logiciel nécessaire à un fonctionnement sûr est donc lié aux conditions de panne du système.

### 2.2.1 Catégories des conditions de panne

Pour une définition complète des catégories de conditions de panne, il y a lieu de se reporter aux règlements et directives applicables, à savoir le document AC 25-1309-1A de la Federal Aviation Administration et/ou le document AMJ 25-1309 des Joint Aviation Authorities, avec leurs mises à jour. Les catégories de conditions de panne énumérées sont établies à partir de ces directives et elles figurent ici dans un souci d'aide à l'utilisation de ce document. Ces catégories sont les suivantes :

a. Catastrophique : Conditions de panne susceptibles d'empêcher la poursuite en toute sécurité d'un vol et d'un atterrissage.

b. Dangereuse : Conditions de panne susceptibles de réduire les possibilités de l'aéronef ou la capacité de l'équipage à faire face à des conditions hostiles pouvant aller jusqu'à :

   (1) une réduction importante des marges de sécurité ou des capacités fonctionnelles.

   (2) des problèmes physiques ou un accroissement de charge de travail tels que l'équipage ne soit plus en mesure d'accomplir sa tâche de manière précise ou complète, ou

   (3) des effets négatifs sur les occupants tels que des blessures graves ou potentiellement mortelles pour un petit nombre d'entre eux.

c. Majeure : Conditions de panne susceptibles de réduire les possibilités de l'aéronef ou la capacité de l'équipage à faire face à des conditions hostiles d'une gravité se traduisant, par exemple, par une réduction significative des marges de sécurité ou des capacités fonctionnelles, un accroissement significatif de la charge de travail de l'équipage ou des conditions affectant son efficacité, ou un inconfort pour les occupants, comportant éventuellement des blessures.

d. Mineure : Conditions de panne n'engendrant pas de réduction significative de la sécurité de l'aéronef, et susceptibles d'entraîner pour l'équipage des actions se situant tout à fait dans le domaine de leurs capacités. Les conditions de panne mineures peuvent comprendre, par exemple, une légère réduction des marges de sécurité ou des capacités fonctionnelles, une légère augmentation de la charge de travail de l'équipage telle que modifications ordinaires de plans de vol, ou une certaine gêne pour les occupants.

e. No Effect: Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.

2.2.2 Software Level Definitions

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

a. Level A: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.

b. Level B: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition for the aircraft.

c. Level C: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.

d. Level D: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.

e. Level E: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.

2.2.3 Software Level Determination

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

*NOTE 1: The applicant may want to consider planned functionality to be added during future developments, as well as potential changes in system requirements allocated to software that may result in a more severe failure condition category and higher software level. It may be desirable to develop the software to a level higher than that determined by the system safety assessment process of the original application, since later development of software life cycle data for substantiating a higher software level application may be difficult.*

*NOTE 2: For airborne systems and equipment mandated by operating regulations, but which do not affect the airworthiness of the aircraft, for example, an accident flight data recorder, the software level needs to be commensurate with the intended function. In some cases, the software level may be specified in equipment minimum performance standards.*

If the anomalous behavior of a software component contributes to more than one failure condition, then the most severe failure condition category of that component determines the software level for that software component. There are various architectural strategies, such as those described in subsection 2.3, which during the evolution of the system design, may result in the software level(s) being revised.

e. Sans effet : Conditions de panne n'affectant pas la capacité opérationnelle de l'aéronef ou n'entraînant pas une augmentation de la charge de travail de l'équipage.

2.2.2 Définition des niveaux logiciels

Le niveau logiciel est fondé sur la contribution du logiciel à d'éventuelles conditions de panne, déterminée par l'analyse de sécurité du système. Le niveau logiciel implique que l'effort nécessaire pour montrer la satisfaction des spécifications de certification varie avec la catégorie de conditions de panne. La définition des niveaux logiciels est :

a. Niveau A : Logiciel dont le dysfonctionnement, mis en évidence par l'analyse de sécurité du système, provoquerait ou contribuerait à une panne d'une fonction du système entraînant une condition de panne "catastrophique" pour l'aéronef.

b. Niveau B : Logiciel dont le dysfonctionnement, mis en évidence par l'analyse de sécurité du système, provoquerait ou contribuerait à une panne d'une fonction du système entraînant une condition de panne "dangereuse" pour l'aéronef.

c. Niveau C : Logiciel dont le dysfonctionnement, mis en évidence par l'analyse de sécurité du système, provoquerait ou contribuerait à une panne d'une fonction du système entraînant une condition de panne "majeure" pour l'aéronef.

d. Niveau D : Logiciel dont le dysfonctionnement, mis en évidence par l'analyse de sécurité du système, provoquerait ou contribuerait à une panne d'une fonction du système entraînant une condition de panne "mineure" pour l'aéronef.

e. Niveau E : Logiciel dont le dysfonctionnement, mis en évidence par l'analyse de sécurité du système, provoquerait ou contribuerait à une panne d'une fonction du système entraînant une condition de panne sans effet sur la capacité opérationnelle de l'aéronef ni sur la charge de travail du pilote. Une fois que l'Autorité de certification a confirmé qu'un logiciel était de niveau E, aucune autre recommandation de ce document n'est applicable.

2.2.3 Détermination du niveau logiciel

Initialement, l'analyse de sécurité du système détermine le(s) niveau(x) logiciel(s) d'un système particulier sans tenir compte de la conception du système. L'impact d'une panne, qu'il s'agisse de la perte de la fonction ou de son fonctionnement erroné, est étudié quand on procède à cette détermination.

*NOTA 1 : Le postulant peut vouloir prendre en considération l'addition d'une fonctionnalité prévue lors de développements ultérieurs, ainsi que des modifications potentielles des spécifications du système allouées au logiciel, ce qui peut avoir pour conséquence une catégorie de condition de panne plus sévère et un niveau logiciel plus élevé. Il peut être souhaitable de développer le logiciel avec un niveau plus élevé que celui déterminé par l'analyse de sécurité du système pour l'application d'origine, en raison de la difficulté possible d'un développement ultérieur du logiciel satisfaisant un niveau logiciel supérieur.*

*NOTA 2 : Pour les systèmes et équipements de bord n'affectant pas la navigabilité de l'aéronef, mais soumis néanmoins à des règles opérationnelles de fonctionnement, par exemple un enregistreur de données de vol, le niveau logiciel doit être cohérent avec la fonction prévue. Dans certains cas, on peut spécifier le niveau logiciel dans les spécifications de performances minimales de l'équipement.*

Si le dysfonctionnement d'un composant logiciel contribue à plus d'une condition de panne, la condition de panne la plus sévère détermine le niveau logiciel de ce composant. Il existe plusieurs stratégies d'architecture, telles que celles décrites à la sous-section 2.3, qui peuvent entraîner, au cours de l'évolution de la conception du système, une révision du (ou des) niveau(x) logiciel(s).

A system function may be allocated to one or more partitioned software components. A parallel implementation is one in which a system function is implemented with multiple software components such that anomalous behavior of more than one component is required to produce the failure condition. For a parallel implementation, at least one software component will have the software level associated with the most severe failure condition category for that system function. The software level for the other components are determined using the failure condition category associated with loss of that function. Examples of such implementations are described in paragraphs 2.3.2, Multiple-Version Dissimilar Software, and 2.3.3, Safety Monitoring.

A serial implementation is one in which multiple software components are used for a system function such that anomalous behavior of any of the components could produce the failure condition. In this implementation, the software components will have the software level associated with the most severe failure condition category of the system function.

Development of software to a software level does not imply the assignment of a failure rate for that software. Thus, software levels or software reliability rates based on software levels cannot be used by the system safety assessment process as can hardware failure rates.

Strategies which depart from the guidelines of this paragraph (2.2.3) need to be justified by the system safety assessment process.

## 2.3 SYSTEM ARCHITECTURAL CONSIDERATIONS

If the system safety assessment process determines that the system architecture precludes anomalous behavior of the software from contributing to the most severe failure condition of a system, then the software level is determined by the most severe category of the remaining failure conditions to which the anomalous behavior of the software can contribute. The system safety assessment process considers the architectural design decisions to determine whether they affect software level or software functionality. Guidance is provided on several architectural strategies that may limit the impact of errors, or detect errors and provide acceptable system responses to contain the errors. These architectural techniques are not intended to be interpreted as the preferred or required solutions.

### 2.3.1 Partitioning

Partitioning is a technique for providing isolation between functionally independent software components to contain and/or isolate faults and potentially reduce the effort of the software verification process. If protection by partitioning is provided, the software level for each partitioned component may be determined using the most severe failure condition category associated with that component.

Guidance for partitioning includes:

a. These aspects of the system should be considered when designing partitioning protection to determine their potential for violating that protection:

   (1) Hardware resources: processors, memory devices, I/O devices, interrupts, and timers.

   (2) Control coupling: vulnerability to external access.

   (3) Data coupling: shared or overlaying data, including stacks and processor registers.

   (4) Failure modes of hardware devices associated with the protection mechanisms.

Une fonction du système peut être allouée à un ou plusieurs composants logiciels partitionnés. Une architecture parallèle est une architecture dans laquelle une fonction du système est réalisée au moyen de composants logiciels multiples, de telle sorte qu'un dysfonctionnement de plus d'un composant est nécessaire pour provoquer la condition de panne. Pour une telle architecture, l'un au moins des composants logiciels aura le niveau logiciel lié à la condition de panne la plus sévère pour la fonction du système. Le niveau logiciel des autres composants est déterminé en considérant la catégorie des conditions de panne liées à la perte de la fonction. Des exemples de telles architectures sont décrites dans les paragraphes 2.3.2 Logiciels à versions multiples dissimilaires et 2.3.3 Surveillance.

Une architecture série est une architecture dans laquelle on utilise des composants logiciels multiples pour une fonction du système, de telle sorte qu'un dysfonctionnement de l'un quelconque des composants peut provoquer la condition de panne. Pour une telle architecture, les composants logiciels auront le niveau logiciel lié à la condition de panne la plus sévère pour la fonction du système.

Le développement d'un logiciel à un niveau donné n'implique pas qu'on lui attribue un taux de panne. De ce fait, l'analyse de sécurité du système ne peut pas utiliser des niveaux logiciels ou des taux de fiabilité du logiciel fondés sur les niveaux logiciels comme on peut le faire pour les taux de panne du matériel.

Les stratégies qui s'écartent des recommandations de ce paragraphe (2.2.3) doivent être justifiées par l'analyse de sécurité du système.

## 2.3 CONSIDERATIONS SUR L'ARCHITECTURE DES SYSTEMES

Si l'analyse de sécurité du système détermine que l'architecture du système exclut la contribution d'un dysfonctionnement du logiciel à la condition de panne la plus sévère, le niveau logiciel est alors déterminé par la condition de panne la plus sévère restante, à laquelle un dysfonctionnement du logiciel peut contribuer. L'analyse de sécurité du système prend en considération les décisions de conception d'architecture afin de déterminer si elles affectent le niveau ou la fonctionnalité du logiciel. Des directives sont fournies ci-dessous sur plusieurs stratégies qui peuvent limiter l'impact des erreurs, ou détecter les défauts et donner des réponses acceptables du système pour confiner les erreurs. Ces techniques architecturales ne doivent pas être interprétées comme des solutions préférées ou imposées.

### 2.3.1 Partitionnement

Le partitionnement est une technique assurant l'isolement entre des composants logiciels fonctionnellement indépendants de manière à contenir et/ou isoler les défauts et réduire potentiellement l'effort de vérification du logiciel. S'il y a protection par partitionnement, le niveau logiciel de chaque composant de la partition peut être déterminé par la catégorie la plus sévère des conditions de panne associées à chaque composant.

Les directives pour le partitionnement comprennent :

a. Les caractéristiques du système à considérer, lors de la conception de la protection de la partition, pour leur potentiel à mettre en défaut cette protection :

   (1) Ressources matérielles : processeurs, mémoires, dispositifs d'entrée/sortie, gestionnaire d'interruptions, horloges.

   (2) Couplage par les flux de contrôle : vulnérabilité vis à vis des accès extérieurs.

   (3) Couplage par les flux de données : données partagées ou se recouvrant, y compris piles et registres de processeurs.

   (4) Modes de panne des dispositifs matériels associés aux mécanismes de protection.

b. The software life cycle processes should address the partitioning design considerations, including the extent and scope of interactions permitted between the partitioned components, and whether the protection is implemented by hardware or by a combination of hardware and software.

c. If the partitioning protection involves software, then that software should be assigned the software level corresponding to the highest level of the partitioned software components.

### 2.3.2 Multiple-Version Dissimilar Software

Multiple-version dissimilar software is a system design technique that involves producing two or more components of software that provide the same function in a way that may avoid some sources of common errors between the components. Multiple-version dissimilar software is also referred to as multi-version software, dissimilar software, N-version programming, or software diversity.

Software life cycle processes completed or activated before dissimilarity is introduced into a development, remain potential error sources. System requirements may specify a hardware configuration which provides for the execution of multiple-version dissimilar software.

The degree of dissimilarity and hence the degree of protection is not usually measurable. Probability of loss of system function will increase to the extent that the safety monitoring associated with dissimilar software versions detects actual errors or experiences transients that exceed comparator

threshold limits. Dissimilar software versions are usually used, therefore, as a means of providing additional protection after the software verification process objectives for the software level, as described in section 6, have been satisfied. Dissimilar software verification methods may be reduced from those used to verify single version software if it can be shown that the resulting potential loss of system function is acceptable as determined by the system safety assessment process.

Verification of multiple-version dissimilar software is discussed in paragraph 12.3.3.

### 2.3.3 Safety Monitoring

Safety monitoring is a means of protecting against specific failure conditions by directly monitoring a function for failures which would contribute to the failure condition. Monitoring functions may be implemented in hardware, software, or a combination of hardware and software.

Through the use of monitoring techniques, the software level of the monitored function may be reduced to the level associated with the loss of its related system function. To allow this level reduction, there are three important attributes of the monitor that should be determined:

a. Software level: Safety monitoring software is assigned the software level associated with the most severe failure condition category for the monitored function.

b. System fault coverage: Assessment of the system fault coverage of a monitor ensures that the monitor's design and implementation are such that the faults which it is intended to detect will be detected under all necessary conditions.

c. Independence of Function and Monitor: The monitor and protective mechanism are not rendered inoperative by the same failure that causes the hazard.

b. Le cycle de vie du logiciel doit définir les caractéristiques du partitionnement, telles que l'importance et la nature des interactions autorisées entre les composants de la partition, et déterminer si la protection est assurée par du matériel ou par une combinaison de matériel et de logiciel.

c. Si la protection de la partition implique du logiciel, celui-ci doit se voir allouer le niveau correspondant au niveau logiciel le plus élevé des composants de la partition.

### 2.3.2 Logiciels à versions multiples dissimilaires

Un logiciel à versions multiples dissimilaires est une technique de conception de système qui implique de réaliser deux ou plusieurs composants logiciels assurant la même fonction d'une manière pouvant éviter certaines sources d'erreurs communes entre les composants. Le logiciel à versions multiples dissimilaires est aussi appelé logiciel multiversions, logiciel dissimilaire, programmation à N-versions, ou diversité de logiciel.

Les processus du cycle de vie du logiciel, achevés ou activés, avant que la dissimilarité ne soit décidée pour un développement, restent des sources potentielles d'erreurs communes. Les spécifications du système peuvent définir une configuration matérielle qui assure l'exécution d'un logiciel à versions multiples dissimilaires.

Le degré de dissimilarité et par suite le degré de protection ne sont pas habituellement mesurables. La probabilité de perte de fonction du système va s'accroître dans la mesure où la surveillance assurée grâce aux versions de logiciel dissimilaire détecte les erreurs réelles ou subit des transitoires qui

dépassent les limites de seuil du comparateur. On utilise ainsi les versions de logiciel dissimilaire comme un moyen de protection supplémentaire une fois que les objectifs du processus de vérification du logiciel, pour ce qui concerne le niveau logiciel décrit dans la section 6, ont été satisfaits. Les méthodes de vérification d'un logiciel dissimilaire peuvent se réduire à celles employées pour la vérification d'un logiciel à version unique si on peut montrer que la perte potentielle de fonction du système qui en résulte est acceptable, conformément au processus d'analyse de sécurité du système.

La vérification des logiciels à versions multiples dissimilaires est traitée dans le paragraphe 12.3.3.

### 2.3.3. Surveillance

La surveillance est un moyen de protection contre des conditions de pannes spécifiques qui consiste à surveiller directement une fonction, pour des pannes qui contribueraient à la condition de panne. Les fonctions de surveillance peuvent être réalisées dans le matériel, dans le logiciel, ou dans une combinaison de matériel et de logiciel.

Grâce à l'utilisation des techniques de surveillance, il est possible de réduire le niveau logiciel de la fonction surveillée au niveau lié à la perte de la fonction du système associée. Pour permettre cette réduction de niveau, il est nécessaire de déterminer trois paramètres importants du dispositif de surveillance :

a. Niveau logiciel : On alloue au logiciel de surveillance le niveau logiciel lié à la catégorie de condition de panne la plus sévère de la fonction surveillée.

b. Couverture des défauts du système : L'évaluation de la couverture des défauts du système par un dispositif de surveillance assure que la conception et la réalisation du dispositif sont telles qu'il détecte dans toutes les situations nécessaires les défauts qu'il a pour mission de détecter.

c. Indépendance de la fonction et du dispositif de surveillance : Le dispositif de surveillance et le mécanisme de protection ne sont pas rendus inopérants par la même panne qui cause le danger.

## 2.4 SYSTEM CONSIDERATIONS FOR USER-MODIFIABLE SOFTWARE, OPTION-SELECTABLE SOFTWARE AND COMMERCIAL OFF-THE-SHELF SOFTWARE

The potential effects of user modification are determined by the system safety assessment process and used to develop the software requirements, and then, the software verification process activities. Designing for user-modifiable software is discussed further in paragraph 5.2.3. A change that affects the non-modifiable software, its protection, or the modifiable software boundaries is a software modification and discussed in paragraph 12.1.1. For this document, a modifiable component is that part of the software that is intended to be changed by the user, and a non-modifiable component is that which is not intended to be changed by the user.

Some airborne systems and equipment may include optional functions which may be selected by software programmed options rather than by hardware connector pins. The option-selectable software functions are used to select a particular configuration within the target computer. See paragraph 5.4.3 for guidelines on deactivated code.

Guidance for system considerations for user-modifiable software, option-selectable software, and commercial off-the-shelf software includes:

a. User-modifiable software: Users may modify software within the modification constraints without certification authority review, if the system requirements provide for user modification.

b. The system requirements should specify the mechanisms which prevent the user modification from affecting system safety whether or not they are correctly implemented. The software which provides the protection for user modification should be at the same software level as the function it is protecting from errors in the modifiable component.

c. If the system requirements do not include provision for user modification, the software should not be modified by the user unless compliance with this document is demonstrated for the modification.

d. At the time of the user modification, the user should take responsibility for all aspects of the user-modifiable software, for example, software configuration management, software quality assurance, and software verification.

e. Option-selectable software: When software programmed options are included, means should be provided to ensure that inadvertent selections involving non-approved configurations for the target computer within the installation environment cannot be made.

f. Commercial off-the-shelf software: COTS software included in airborne systems or equipment should satisfy the objectives of this document.

g. If deficiencies exist in the software life cycle data of COTS software, the data should be augmented to satisfy the objectives of this document. The guidelines in paragraphs 12.1.4, Upgrading A Development Baseline, and 12.3.5, Product Service History, may be relevant in this instance.

## 2.5 SYSTEM DESIGN CONSIDERATIONS FOR FIELD-LOADABLE SOFTWARE

Field-loadable airborne software refers to software or data tables that can be loaded without removing the system or equipment from its installation. The safety-related requirements associated with the software data loading function are part of the system requirements. If the inadvertent enabling of the software data loading function could induce a system failure condition, a safety-related requirement for the software data loading function is specified in the system requirements.

## 2.4 CONSIDERATIONS RELATIVES AU SYSTEME POUR LES LOGICIELS MODIFIABLES PAR L'UTILISATEUR, LES LOGICIELS A OPTION, ET LES LOGICIELS DU COMMERCE SUR ETAGERE

Les effets potentiels des modifications par l'utilisateur sont déterminés par l'analyse de sécurité du système. Ils sont utilisés pour développer les spécifications du logiciel et, de ce fait, les activités du processus de vérification. La conception des logiciels modifiables par l'utilisateur est abordée plus loin dans le paragraphe 5.2.3. Une modification affectant un logiciel non modifiable, sa protection, ou les limites d'un logiciel modifiable, constitue une modification de logiciel et est traitée dans le paragraphe 12.1.1. Dans l'esprit de ce document, un composant modifiable est une partie de logiciel conçue pour pouvoir être modifiée par l'utilisateur, et un composant non modifiable est une partie de logiciel qui n'est pas conçue pour pouvoir être modifiée par l'utilisateur.

Certains systèmes et équipements de bord peuvent comporter des fonctions optionnelles qui peuvent être sélectées au moyen d'options programmées dans le logiciel plutôt que par des broches de connecteurs. On utilise les fonctions de logiciels à option pour choisir une configuration particulière de la machine cible. Voir le paragraphe 5.4.3 pour les recommandations relatives au code désactivé.

Les directives pour les logiciels modifiables par l'utilisateur, les logiciels à option, et les logiciels du commerce sur étagère sont les suivantes :

a. Logiciel modifiable par l'utilisateur : Les utilisateurs peuvent modifier le logiciel dans les limites établies sans examen par l'Autorité de certification, si les spécifications du système autorisent les modifications par l'utilisateur.

b. Les spécifications du système doivent définir les mécanismes permettant d'éviter qu'une modification par l'utilisateur puisse affecter la sécurité du système, que cette modification soit correctement réalisée ou non. Le logiciel assurant la protection vis à vis des modifications par l'utilisateur doit avoir le même niveau logiciel que la fonction du composant modifiable qu'il protège contre les erreurs.

c. Si les spécifications du système ne prévoient pas les modifications par l'utilisateur, ce dernier ne peut pas modifier le logiciel sauf s'il démontre la conformité de la modification avec ce document.

d. Au moment de la modification par l'utilisateur, ce dernier doit en prendre la responsabilité pour tous les aspects du logiciel modifiable par l'utilisateur, par exemple la gestion de configuration du logiciel, l'assurance qualité du logiciel, et la vérification du logiciel.

e. Logiciel sélectable par option : Quand des options programmées sont inclues dans le logiciel, des moyens doivent exister pour que l'on ne puisse pas effectuer par inadvertance des sélections impliquant pour la machine cible des configurations non approuvées dans l'environnement opérationnel.

f. Logiciels du commerce sur étagère (COTS) : Les logiciels de ce type inclus dans les systèmes ou équipements de bord doivent satisfaire les objectifs de ce document.

g. S'il existe des insuffisances dans les données du cycle de vie des logiciels du commerce sur étagère, il est nécessaire de compléter ces données de manière à satisfaire les objectifs de ce document. Les recommandations des paragraphes 12.1.4, Amélioration du Référentiel de Développement, et 12.3.5, Historique du Produit en Service, peuvent être utiles à cet égard.

## 2.5 CONSIDERATION DE CONCEPTION DES SYSTEMES POUR LES LOGICIELS TELECHARGEABLES

System safety considerations relating to field-loadable software include:

- Detection of corrupted or partially loaded software.
- Determination of the effects of loading the inappropriate software.
- Hardware/software compatibility.
- Software/software compatibility.
- Aircraft/software compatibility.
- Inadvertent enabling of the field loading function.
- Loss or corruption of the software configuration identification display.

Guidance for field-loadable software includes:

a. Unless otherwise justified by the system safety assessment process, the detection mechanism for partial or corrupted software loads should be assigned the same failure condition or software level as the most severe failure condition or software level associated with the function that uses the software load.

b. If a system has a default mode when inappropriate software or data is loaded, then each partitioned component of the system should have safety-related requirements specified for operation in this mode which address the potential failure condition.

c. The software loading function, including support systems and procedures, should include a means to detect incorrect software and/or hardware and/or aircraft combinations and should provide protection appropriate to the failure condition of the function.

d. If software is part of an airborne display mechanism that is the means for ensuring that the aircraft conforms to a certified configuration, then that software should either be developed to the highest level of the software to be loaded, or the system safety assessment process should justify the integrity of an end-to-end check of the software configuration identification.

## 2.6 SYSTEM REQUIREMENTS CONSIDERATIONS FOR SOFTWARE VERIFICATION

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

a. The system requirements for airborne software establish two characteristics of the software:

   (1) The software performs specified functions as defined by the system requirements.

   (2) The software does not exhibit specific anomalous behavior(s) as determined by the system safety assessment process. Additional system requirements are generated to eliminate the anomalous behavior.

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

Les logiciels embarqués téléchargeables sont des logiciels ou des tableaux de données qu'on peut charger sans démontage du système ou de l'équipement. Les exigences liées à la sécurité, associées à la fonction de chargement des données du logiciel, constituent une partie des spécifications du système. Si, l'activation, par inadvertance, de la fonction de chargement des données du logiciel était susceptible d'induire une condition de panne du système, les spécifications du système comporteraient une exigence de sécurité pour la fonction de chargement.

Les aspects de sécurité du système liés aux logiciels téléchargeables comprennent :

- La détection des logiciels altérés ou partiellement chargés.
- La détermination des effets du chargement d'un logiciel non approprié.
- La compatibilité matériel/logiciel.
- La compatibilité logiciel/logiciel.
- La compatibilité aéronef/logiciel.
- L'activation par inadvertance de la fonction de téléchargement.
- La perte ou altération de l'affichage de l'identification de la configuration du logiciel.

Les directives pour les logiciels téléchargeables sont les suivantes :

a. Sauf si c'est justifié par ailleurs par l'analyse de sécurité du système, le mécanisme de détection du chargement d'un logiciel partiel ou altéré doit se voir allouer la même condition de panne ou le même niveau logiciel que la condition de panne ou le niveau logiciel les plus sévères liés à la fonction utilisant le téléchargement du logiciel.

b. Si un système a un mode par défaut en cas de téléchargement de logiciel ou de données non appropriés, chaque composant du partitionnement du système doit être soumis, pour le fonctionnement dans ce mode, à des exigences de sécurité tenant compte de la condition de panne potentielle.

c. La fonction de chargement du logiciel, y compris les systèmes et procédures associés, doit comporter un moyen de détecter des combinaisons incorrectes de logiciel et/ou de matériel et/ou d'aéronef et doit fournir une protection appropriée à la condition de panne de la fonction.

d. Si le logiciel fait partie d'un mécanisme embarqué d'affichage utilisé comme le moyen d'assurer que l'aéronef est conforme à une configuration homologuée, le logiciel doit alors être développé au niveau le plus élevé du logiciel à charger, ou bien l'analyse de sécurité du système doit justifier de l'intégrité d'un contrôle de bout-en-bout de l'identification de la configuration du logiciel.

## 2.6 CONSIDERATIONS DE SYSTEME POUR LA VERIFICATION DES LOGICIELS

Les spécifications du système sont établies à partir d'exigences opérationnelles et d'exigences liées à la sécurité, qui résultent de l'analyse de sécurité du système. Les considérations suivantes sont à considérer :

a. Les spécifications du système pour les logiciels embarqués établissent deux caractéristiques du logiciel :

   (1) Le logiciel assure des fonctions définies comme indiqué par les spécifications du système.

   (2) Le logiciel ne manifeste pas un (ou des) comportement(s) anormal (anormaux), déterminé(s) par l'analyse de sécurité du système. Dans ce cas des spécifications du système complémentaires sont définies afin d'éliminer le dysfonctionnement.

b. Ces spécifications du système sont exprimées sous la forme d'exigences de haut niveau du logiciel qui sont vérifiées par les activités du processus de vérification du logiciel.

## 2.7 SOFTWARE CONSIDERATIONS IN SYSTEM VERIFICATION

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

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

## 2.7 CONSIDERATIONS RELATIVES AU LOGICIEL DANS LA VERIFICATION DES SYSTEMES

Les directives pour la vérification des systèmes n'entrent pas dans le cadre de ce document. Cependant, le cycle de vie du logiciel aide le processus de vérification du système et est en interaction avec lui. Il est nécessaire de connaître les détails de la conception du logiciel liés à la fonctionnalité du système pour aider à la vérification du système.

La vérification du système peut apporter une couverture significative de la structure du code. On peut utiliser l'analyse de couverture des essais de vérification du système pour atteindre les objectifs de couverture de diverses activités de test décrites dans la vérification du logiciel.

# SECTION 3

# SOFTWARE LIFE CYCLE

This section discusses the software life cycle processes, software life cycle definition, and transition criteria between software life cycle processes. The guidelines of this document do not prescribe a preferred software life cycle, but describe the separate processes that comprise most life cycles and the interactions between them. The separation of the processes is not intended to imply a structure for the organization(s) that perform them. For each software product, the software life cycle(s) is constructed that includes these processes.

## 3.1 SOFTWARE LIFE CYCLE PROCESSES

The software life cycle processes are:

- The software planning process that defines and coordinates the activities of the software development and integral processes for a project. Section 4 describes the software planning process.
- The software development processes that produce the software product. These processes are the software requirements process, the software design process, the software coding process, and the integration process. Section 5 describes the software development processes.
- The integral processes that ensure the correctness, control, and confidence of the software life cycle processes and their outputs. The integral processes are the software verification process, the software configuration management process, the software quality assurance process, and the certification liaison process. It is important to understand that the integral processes are performed concurrently with the software development processes throughout the software life cycle. Sections 6 through 9 describe the integral processes.

## 3.2 SOFTWARE LIFE CYCLE DEFINITION

A project defines one or more software life cycle(s) by choosing the activities for each process, specifying a sequence for the activities, and assigning responsibilities for the activities.

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

Figure 3-1 illustrates the sequence of software development processes for several components of a single software product with different software life cycles. Component W implements a set of system requirements by developing the software requirements, using those requirements to define a software design, implementing that design into source code, and then integrating the component into the hardware. Component X illustrates the use of previously developed software used in a certified aircraft or engine. Component Y illustrates the use of a simple, partitioned function that can be coded directly from the software requirements. Component Z illustrates the use of a prototyping strategy. Usually, the goals of prototyping are to better understand the software requirements and to mitigate development and technical risks. The initial requirements are used as the basis to implement a prototype. This prototype is evaluated in an environment representative of the intended use of the system under development. Results of the evaluation are used to refine the requirements.

# SECTION 3

# CYCLE DE VIE DU LOGICIEL

Cette section traite des processus du cycle de vie du logiciel, de la définition du cycle de vie du logiciel, et des critères de transition entre les processus du cycle de vie du logiciel. Les recommandations de ce document n'imposent pas de préférence pour un cycle de vie de logiciel, mais décrivent les processus séparés que comprennent la plupart des cycles de vie ainsi que leurs interactions mutuelles. La séparation des processus n'implique pas de structure pour l' (ou les) organisation(s) qui les réalisent. Pour chaque logiciel, un cycle de vie du logiciel incluant ces processus est élaboré.

## 3.1 PROCESSUS DU CYCLE DE VIE DU LOGICIEL

Les processus du cycle de vie d'un logiciel sont les suivants :

- La planification du logiciel, qui définit et coordonne les activités de développement du logiciel et des processus intégraux d'un projet. La section 4 décrit la planification des logiciels.
- Les processus de développement du logiciel, qui fournissent le produit logiciel. Ces processus sont la définition des spécifications du logiciel, la conception du logiciel, le codage du logiciel, et l'intégration. La section 5 décrit les processus de développement des logiciels.
- Les processus intégraux, qui assurent l'exactitude, la qualité de gestion, et le degré de confiance des processus du cycle de vie du logiciel et de leurs produits. Les processus intégraux sont le processus de vérification du logiciel, le processus de gestion de configuration du logiciel, le processus d'assurance qualité du logiciel, et le processus de coordination pour la certification. Il est important de comprendre que les processus intégraux sont mis en oeuvre en même temps que le développement du logiciel pendant tout le cycle de vie du logiciel. Les sections 6 à 9 décrivent les processus intégraux.

## 3.2 DEFINITION DU CYCLE DE VIE D'UN LOGICIEL

Un projet définit un ou plusieurs cycle(s) de vie du logiciel en choisissant les activités pour chaque processus, en spécifiant le séquencement des activités, et en allouant des responsabilités pour les activités.

Pour un projet spécifique, l'enchaînement de ces processus est déterminé par les paramètres du projet, tels que fonctionnalité et complexité du système, taille et complexité du logiciel, stabilité des spécifications, utilisation de produits développés antérieurement, stratégies de développement, et disponibilité de matériel. L'enchaînement habituel des processus de développement d'un logiciel est : spécifications, conception, codage, et intégration.

La Figure 3.1 illustre l'enchaînement des processus de développement d'un logiciel pour plusieurs composants d'un produit logiciel unique avec différents cycles de vie du logiciel. Le composant W réalise un ensemble de spécifications du système en développant des spécifications du logiciel, en utilisant ces spécifications pour définir une conception du logiciel, en réalisant cette conception dans un Code Source, puis en intégrant le composant dans le

matériel. Le composant X illustre l'utilisation d'un logiciel développé antérieurement, employé sur un aéronef ou un moteur déjà certifié. Le composant Y illustre l'utilisation d'une fonction partitionnée qu'on peut coder directement à partir des spécifications du logiciel. Le composant Z illustre l'utilisation d'une stratégie d'élaboration de prototype. Habituellement, les buts de la création d'un prototype sont de mieux comprendre les spécifications du logiciel et d'atténuer l'effort développement et les risques techniques. Les spécifications initiales sont utilisées comme base de réalisation d'un prototype. Ce prototype est évalué dans un environnement représentatif de l'utilisation prévue du système en cours de développement. Les produits de cette évaluation sont utilisés pour affiner les spécifications.

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

The various parts of the selected software life cycle are tied together with a combination of incremental integration process and software verification process activities.

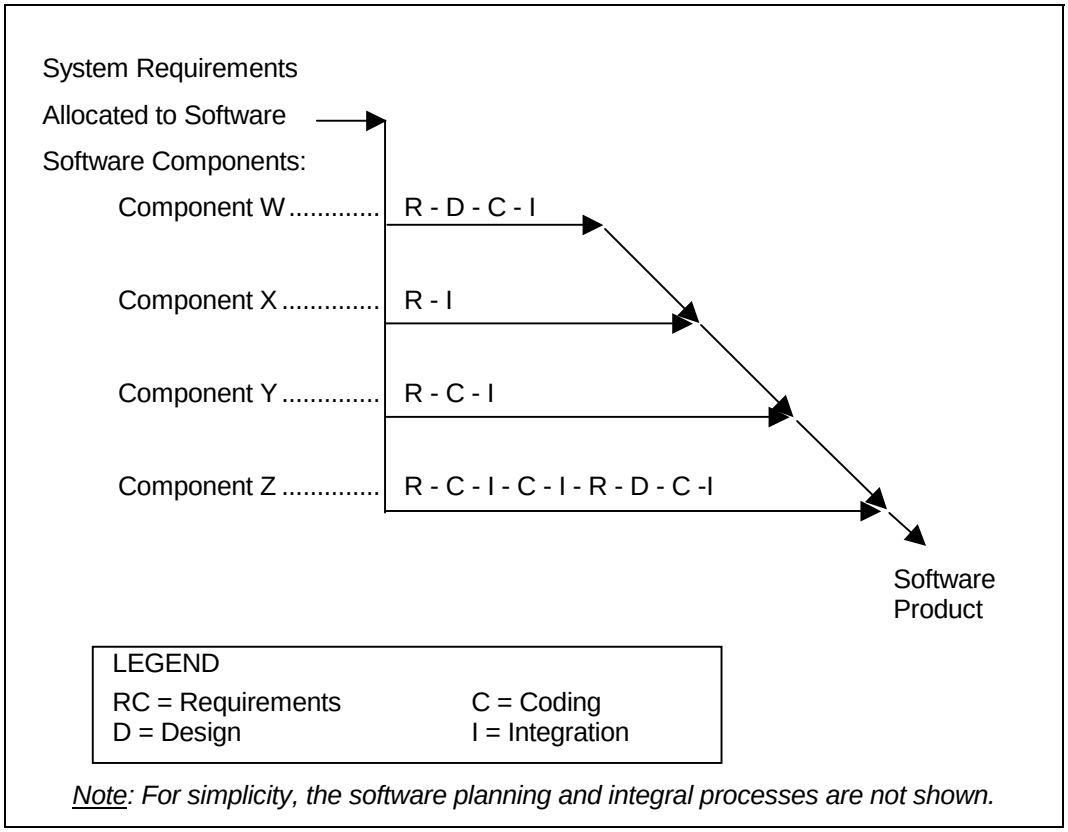
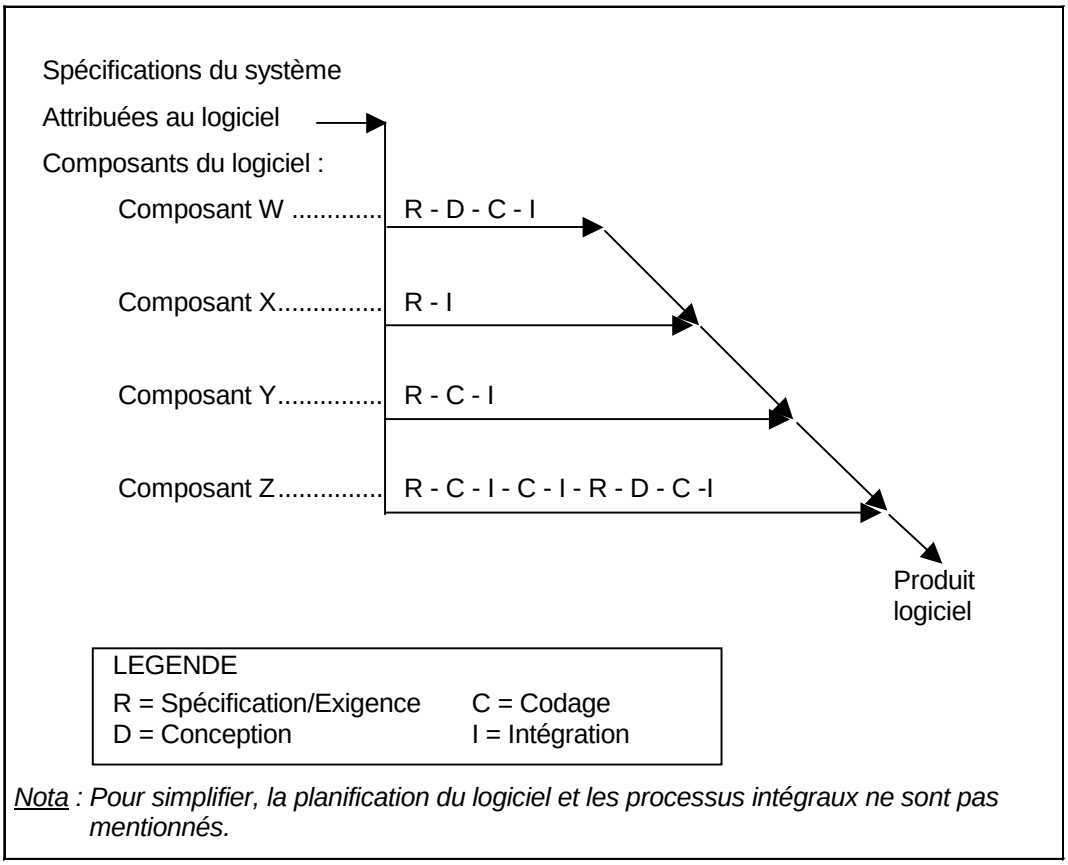



FIGURE 3-1: EXAMPLE OF A SOFTWARE PROJECT USING FOUR DIFFERENT DEVELOPMENT SEQUENCES

Les processus du cycle de vie d'un logiciel peuvent être itératifs, c'est-à-dire engagés et réengagés. La chronologie et le degré d'itération varient selon le développement incrémental des fonctions du système, de sa complexité, du développement de ses spécifications, de la disponibilité du matériel, des effets de retour sur des processus antérieurs, et d'autres paramètres du projet.

Les diverses parties du cycle de vie d'un logiciel choisi sont liées les unes aux autres par une combinaison d'activités incrémentales du processus d'intégration et de vérification de logiciel.




*Nota : Pour simplifier, la planification du logiciel et les processus intégraux ne sont pas mentionnés.*

FIGURE 3-1 : EXEMPLE D'UN PROJET DE LOGICIEL UTILISANT QUATRE SEQUENCES DIFFERENTES DE DEVELOPPEMENT

## 3.3 TRANSITION CRITERIA BETWEEN PROCESSES

Transition criteria are used to determine whether a process may be entered or re-entered. Each software life cycle process performs activities on inputs to produce outputs. A process may produce feedback to other processes and receive feedback from others. Definition of feedback includes how information is recognized, controlled and resolved by the receiving process. An example of a feedback definition is problem reporting.

The transition criteria will depend on the planned sequence of software development processes and integral processes, and may be affected by the software level. Examples of transition criteria which may be chosen are: that the software verification process reviews have been performed; the input is an identified configuration item; and a traceability analysis has been completed for the input.

Every input to a process need not be complete before that process can be initiated, if the transition criteria established for the process are satisfied. Guidance includes:

a. If a process acts on partial inputs, subsequent inputs to the process should be examined to determine that the previous outputs of the software development and software verification processes are still valid.

## 3.3 CRITERES DE TRANSITION ENTRE PROCESSUS

On utilise les critères de transition pour déterminer si un processus peut être engagé ou réengagé. Chaque processus du cycle de vie d'un logiciel réalise des activités sur des données d'entrée afin de produire des sorties. Un processus peut produire sur d'autres processus et recevoir lui-même des informations en retour. La définition d'une information en retour comprend la manière dont l'information est identifiée, contrôlée, et traitée par le processus de réception. Le compte rendu d'anomalies en constitue un exemple.

Les critères de transition dépendent de l'enchaînement planifié des processus de développement du logiciel, et peuvent être affectés par le niveau logiciel. Des exemples de critères de transition sont : les revues des processus de vérification du logiciel ont bien été réalisées, les données d'entrée correspondent bien à un élément de configuration identifié, une analyse de traçabilité des données d'entrée a bien été faite.

Il n'est pas nécessaire que chaque donnée d'entrée d'un processus soit complète avant l'initialisation de ce processus si les critères de transition établis pour ce processus sont satisfaits. Dans ce cas les directives sont les suivantes :

a. Si un processus agit sur des données d'entrée partielles, on doit examiner les données d'entrée à venir du processus en vue de déterminer si les produits antérieurs du développement du logiciel et de sa vérification sont toujours valables.

# SECTION 4

# SOFTWARE PLANNING PROCESS

This section discusses the objectives and activities of the software planning process. This process produces the software plans and standards that direct the software development processes and the integral processes. Table A-1 of Annex A is a summary of the objectives and outputs of the software planning process by software level.

## 4.1 SOFTWARE PLANNING PROCESS OBJECTIVES

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

a. The activities of the software development processes and integral processes of the software life cycle that will address the system requirements and software level(s) are defined (subsection 4.2).

b. The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria are determined (section 3).

c. The software life cycle environment, including the methods and tools to be used for the activities of each software life cycle process have been selected (subsection 4.4).

d. Additional considerations, such as those discussed in section 12, have been addressed, if necessary.

e. Software development standards consistent with the system safety objectives for the software to be produced are defined (subsection 4.5).

f. Software plans that comply with subsection 4.3 and section 11 have been produced.

g. Development and revision of the software plans are coordinated (subsection 4.3).

## 4.2 SOFTWARE PLANNING PROCESS ACTIVITIES

Effective planning is a determining factor in producing software that satisfies the guidelines of this document. Guidance for the software planning process includes:

a. The software plans should be developed at a point in time in the software life cycle that provides timely direction to the personnel performing the software development processes and integral processes. See also the guidelines of subsection 9.1.

b. The software development standards to be used for the project should be defined or selected.

c. Methods and tools should be chosen that provide error prevention in the software development processes.

d. The software planning process should provide coordination between the software development and integral processes to provide consistency among strategies in the software plans.

# SECTION 4

# PLANIFICATION DU LOGICIEL

Cette section traite des objectifs et des activités de planification du logiciel. Ce processus fournit les plans et les règles, qui orientent le développement du logiciel et les processus intégraux. Le Tableau A-1 de l'Annexe A constitue un résumé des objectifs et des produits du processus de planification du logiciel en fonction du niveau logiciel.

## 4.1 OBJECTIFS DE LA PLANIFICATION DU LOGICIEL

La planification du logiciel a pour but de définir les moyens de réalisation du logiciel satisfaisant les spécifications du système et donnant la confiance nécessaire pour répondre aux exigences de navigabilité. Les objectifs de la planification du logiciel sont les suivants :

a. La définition du processus de développement du logiciel et des processus intégraux du cycle de vie du logiciel tenant compte des spécifications du système et du (ou des) niveau(x) logiciel(s) (sous-section 4.2).

b. La détermination du (ou des) cycle(s) de vie du logiciel, comprenant les relations mutuelles entre les processus, leur séquencement, les mécanismes de retour d'information, et les critères de transition (Section 3).

c. Le choix de l'environnement du cycle de vie du logiciel, comprenant les méthodes et les outils à utiliser pour les activités de chaque processus du cycle de vie du logiciel (sous-section 4.4).

d. Des considérations complémentaires, telles que celles traitées dans la section 12, si nécessaire.

e. La définition de règles de développement du logiciel, cohérentes avec les objectifs de sécurité du système (sous-section 4.5).

f. L'établissement de plans en accord avec la sous-section 4.3 et la section 11.

g. La coordination du développement et de la révision des plans (sous-section 4.3).

## 4.2 ACTIVITES DE PLANIFICATION DU LOGICIEL

L'efficacité de la planification est un facteur déterminant pour que la réalisation d'un logiciel satisfasse les recommandations de ce document. Les directives pour la planification du logiciel sont les suivantes :

a. Les plans doivent être développés à un moment donné du cycle de vie, de telle sorte qu'ils fournissent en temps voulu les orientations nécessaires au personnel chargé du développement du logiciel et des processus intégraux. Voir aussi les recommandations de la sous-section 9.1.

b. Les règles de développement du logiciel qui seront utilisées pour le projet doivent être définies ou choisies.

c. Les méthodes et outils choisis doivent prévenir l'introduction d'erreurs dans les processus de développement du logiciel.

d. La planification doit permettre de coordonner le développement du logiciel et les processus intégraux de manière à garantir la cohérence des stratégies dans les plans.

e. The software planning process should include a means to revise the software plans as a project progresses.

f. When multiple-version dissimilar software is used in a system, the software planning process should choose the methods and tools to achieve the error avoidance or detection necessary to satisfy the system safety objectives.

g. For the software planning process to be complete, the software plans and software development standards should be under change control and reviews of them completed (subsection 4.6).

h. If deactivated code is planned (subsection 2.4), the software planning process should describe how the deactivated code (selected options, flight test) will be defined, verified and handled to achieve system safety objectives.

i. If user-modifiable code is planned, the process, tools, environment, and data items substantiating the guidelines of paragraph 5.2.3 should be specified in the software plans and standards.

Other software life cycle processes may begin before completion of the software planning process if the plans and procedures for the specific process activity are available.

## 4.3 SOFTWARE PLANS

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

- The Plan for Software Aspects of Certification (subsection 11.1) serves as the primary means for communicating the proposed development methods to the certification authority for agreement, and defines the means of compliance with this document.
- The Software Development Plan (subsection 11.2) defines the software life cycle(s) and software development environment.
- The Software Verification Plan (subsection 11.3) defines the means by which the software verification process objectives will be satisfied.
- The Software Configuration Management Plan (subsection 11.4) defines the means by which the software configuration management process objectives will be satisfied.
- The Software Quality Assurance Plan (subsection 11.5) defines the means by which the software quality assurance process objectives will be satisfied.

Guidance for the software plans includes:

a. The software plans should comply with this document.

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

   (1) The inputs to the process, including feedback from other processes.

   (2) Any integral process activities that may be required to act on these inputs.

   (3) Availability of tools, methods, plans and procedures.

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

e. La planification doit prévoir un moyen de révision des plans au cours du déroulement du projet.

f. Dans le cas de l'utilisation d'un logiciel à versions multiples dissimilaires, la planification doit choisir les méthodes et les outils permettant d'éviter ou de détecter les erreurs, en accord avec les objectifs de sécurité du système.

g. Pour que la planification soit complète, les plans et règles de développement du logiciel doivent être sous gestion de modification et leurs revues doivent être achevées (sous-section 4.6).

h. S'il est prévu du code désactivé (sous-section 2.4), la planification doit décrire la manière dont le code désactivé (options choisies, essai en vol) sera défini, vérifié, et traité pour que les objectifs de sécurité du système soient satisfaits.

i. S'il est prévu du code modifiable par l'utilisateur, le processus, les outils, l'environnement, et les éléments de données répondant aux recommandations du paragraphe 5.2.3 doivent être spécifiés dans les plans et règles de développement du logiciel.

D'autres processus du cycle de vie du logiciel peuvent commencer avant l'achèvement de la planification, si les plans et les procédures de ces processus sont disponibles.

## 4.3 LES PLANS

Les plans ont pour but de définir les moyens de satisfaire les objectifs de ce document. Ils précisent l'organisation qui réalisera ces activités. Les plans sont les suivants :

- Le Plan des Aspects Logiciels de la Certification (sous-section 11.1) sert de support initial pour communiquer les méthodes proposées de développement à l'accord de l'Autorité de certification, et définit les moyens de conformité avec ce document.
- Le Plan de Développement du Logiciel (sous-section 11.2) définit le(s) cycle(s) de vie du logiciel et son environnement de développement.
- Le Plan de Vérification du Logiciel (sous-section 11.3) définit les moyens grâce auxquels les objectifs du processus de vérification du logiciel seront satisfaits.
- Le Plan de Gestion de Configuration du Logiciel (sous-section 11.4) définit les moyens grâce auxquels les objectifs de gestion de configuration du logiciel seront satisfaits.
- Le Plan d'Assurance Qualité du Logiciel (sous-section 11.5) définit les moyens grâce auxquels les objectifs du processus d'assurance qualité du logiciel seront satisfaits.

Les directives pour les plans sont les suivantes :

a. Les plans doivent être conformes à ce document.

b. Les plans doivent définir les critères de transition entre les cycles de vie du logiciel en spécifiant :

   (1) Les données d'entrée du processus, y compris les retours d'information d'autres processus.

   (2) Toutes les activités de processus intégraux qui peuvent être nécessaires pour agir sur ces données d'entrée.

   (3) La disponibilité des outils, méthodes, plans, et procédures.

c. Les plans doivent préciser les procédures à utiliser pour réaliser des modifications de logiciel avant utilisation sur un aéronef ou un moteur certifié. De telles modifications peuvent être la conséquence de retours d'information d'autres processus et peuvent entraîner une modification des plans.

## 4.4 SOFTWARE LIFE CYCLE ENVIRONMENT PLANNING

The purpose of the planning for the software life cycle environment is to define the methods, tools, procedures, programming languages and hardware that will be used to develop, verify, control and produce the software life cycle data (section 11) and software product. Examples of how the software environment chosen can have a beneficial effect on the airborne software include enforcing standards, detecting errors, and implementing error prevention and fault tolerance methods. The software life cycle environment is a potential error source that can contribute to failure conditions. Composition of this software life cycle environment may be influenced by the safety-related requirements determined by the system safety assessment process, for example, the use of dissimilar, redundant components.

The goal of error prevention methods is to avoid errors during the software development processes that might contribute to a failure condition. The basic principle is to choose requirements development and design methods, tools, and programming languages that limit the opportunity for introducing errors, and verification methods that ensure that errors introduced are detected. The goal of fault tolerance methods is to include safety features in the software design or Source Code to ensure that the software will respond correctly to input data errors and prevent output and control errors. The need for error prevention or fault tolerance methods is determined by the system requirements and the system safety assessment process.

The considerations presented above may affect:

- The methods and notations used in the software requirements process and software design process.
- The programming language(s) and methods used in the software coding
- The software development environment tools.
- The software verification and software configuration management tools.
- The need for tool qualification (subsection 12.2).

### 4.4.1 Software Development Environment

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

a. During the software planning process, the software development environment should be chosen to minimize its potential risk to the final airborne software.

b. The use of qualified tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together.

c. Software verification process activities or software development standards, which include consideration of the software level, should be defined to minimize potential software development environment-related errors.

d. If certification credit is sought for use of the tools in combination, the sequence of operation of the tools should be specified in the appropriate plan.

## 4.4 PLANIFICATION DE L'ENVIRONNEMENT DU CYCLE DE VIE D'UN LOGICIEL

La planification de l'environnement du cycle de vie d'un logiciel a pour but de définir les méthodes, les outils, les procédures, les langages de programmation, et le matériel qui seront utilisés pour développer, vérifier, gérer en configuration, et produire les données du cycle de vie du logiciel (section 11) et le produit logiciel. Les exemples suivants qui montrent en quoi le choix de l'environnement d'un logiciel peut avoir un effet bénéfique sur le logiciel embarqué comprennent le renforcement des règles, la détection des erreurs, et la mise en oeuvre de méthodes de prévention des erreurs et de tolérance aux défauts. L'environnement du cycle de vie d'un logiciel est une source potentielle d'erreurs qui peut contribuer à des conditions de panne. La composition de cet environnement peut être influencée par les exigences de sécurité, qui sont elles-mêmes déterminées par l'analyse de sécurité du système, comme par exemple l'utilisation de composants dissimilaires ou redondants.

Les méthodes de prévention des erreurs ont pour but, au cours du développement du logiciel, d'éviter les erreurs qui pourraient contribuer à une condition de panne. Le principe fondamental consiste à choisir des méthodes de développement et de conception des spécifications, des outils, et des langages de programmation qui limitent la probabilité d'introduction d'erreurs, et des méthodes de vérification qui garantissent la détection des erreurs. Les méthodes de tolérance aux défauts ont pour but d'inclure des dispositifs de sécurité dans la conception du logiciel ou dans le Code Source de manière à garantir que le logiciel répondra correctement à des erreurs sur les données d'entrée et évitera les erreurs sur les sorties et les commandes. Le besoin de méthodes de prévention des erreurs et de tolérance aux défauts est déterminé par les spécifications du système et l'analyse de sécurité du système.

Les considérations présentées ci-dessus peuvent affecter :

- Les méthodes et les notations utilisées dans les spécifications et la conception du logiciel.
- Le(s) langage(s) de programmation et les méthodes employées pour le codage du logiciel.
- Les outils de l'environnement de développement du logiciel.
- Les outils de vérification et de gestion de configuration du logiciel.
- La nécessité d'une qualification d'outil (sous-section 12.2).

### 4.4.1 Environnement de développement du logiciel

L'environnement de développement du logiciel est un paramètre important pour la réalisation d'un logiciel de qualité. L'environnement de développement du logiciel peut aussi affecter défavorablement la réalisation d'un logiciel embarqué de plusieurs manières. Par exemple, un compilateur peut introduire des erreurs en délivrant une sortie altérée et un éditeur de liens peut ne pas révéler l'existence d'une erreur d'allocation de mémoire. Les directives pour le choix des méthodes et des outils de l'environnement de développement du logiciel sont les suivantes :

a. Au cours du processus de planification du logiciel, on doit choisir son environnement de développement de manière à minimiser sa contribution aux risques potentiels dans le logiciel embarqué final.

b. On doit choisir l'utilisation d'outils ou combinaisons d'outils et d'éléments de l'environnement de développement du logiciel qualifiés, de manière à obtenir la certitude nécessaire qu'une erreur introduite par un élément sera détectée par un autre. On obtient un environnement acceptable quand tous ses éléments sont utilisés deux-à-deux de façon cohérente.

c. On doit définir les activités du processus de vérification du logiciel ou ses règles de développement, en tenant compte du niveau logiciel, de manière à minimiser les erreurs potentielles de développement du logiciel liées à l'environnement.

d. Dans le cas ou l'on recherche un crédit de certification pour l'utilisation d'une combinaison d'outils, on doit préciser la séquence d'utilisation de ces outils dans le plan approprié.

e. If optional features of software development tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan.

*NOTE: This is especially important where the tool directly generates part of the software product. In this context, compilers are probably the most important tools to consider.*

4.4.2 Language and Compiler Considerations

Upon successful completion of verification of the software product, the compiler is considered acceptable for that product. For this to be valid, the software verification process activities need to consider particular features of the programming language and compiler. The software planning process considers these features when choosing a programming language and planning for verification. Guidance includes:

a. Some compilers have features intended to optimize performance of the object code. If the test cases give coverage consistent with the software level, the correctness of the optimization need not be verified. Otherwise, the impact of these features on structural coverage analysis should be determined following the guidelines of subparagraph 6.4.4.2.

b. To implement certain features, compilers for some languages may produce object code that is not directly traceable to the source code, for example, initialization, built-in error detection or exception handling (subparagraph 6.4.4.2, item b). The software planning process should provide a means to detect this object code and to ensure verification coverage and define the means in the appropriate plan.

c. If a new compiler, linkage editor or loader version is introduced, or compiler options are changed during the software life cycle, previous tests and coverage analyses may no longer be valid. The verification planning should provide a means of reverification which is consistent with the guidelines of section 6 and paragraph 12.1.3.

4.4.3 Software Test Environment

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

a. The emulator or simulator may need to be qualified as described in subsection 12.2.

b. The differences between the target computer and the emulator or simulator, and the effects of those differences on the ability to detect errors and verify functionality, should be considered. Detection of those errors should be provided by other software verification process activities and specified in the Software Verification Plan.

4.5 SOFTWARE DEVELOPMENT STANDARDS

The purpose of the software development standards is to define the rules and constraints for the software development processes. The software development standards include the Software Requirements Standards, the Software Design Standards and the Software Code Standards. The software verification process uses these standards as a basis for evaluating the compliance of actual outputs of a process with intended outputs. Guidance for software development standards includes:

a. The software development standards should comply with section 11.

e. Si des caractéristiques optionnelles d'un outil de développement du logiciel sont utilisées pour un projet, on doit examiner et préciser les effets de ces options dans le plan approprié.

*NOTA : Ce point est particulièrement important quand l'outil génère directement une partie du produit logiciel. Dans ce contexte, les compilateurs constituent probablement les outils les plus importants à prendre en considération*.

4.4.2 Considérations relatives aux langages et aux compilateurs

Lorsque la vérification du produit logiciel s'achève de manière satisfaisante, le compilateur est considéré comme acceptable pour ce produit. Pour que ceci soit valable, il est nécessaire que les activités du processus de vérification du logiciel tiennent compte des caractéristiques particulières du langage de programmation et du compilateur. La planification du logiciel tient compte de ces caractéristiques lors du choix d'un langage de programmation et de l'établissement du plan de vérification. Les directives sont les suivantes :

a. Certains compilateurs possèdent des caractéristiques destinées à optimiser les performances du code objet. Si les jeux de test donnent une couverture cohérente avec le niveau logiciel, il n'est pas nécessaire de vérifier que l'optimisation est correcte. Dans le cas contraire, on doit déterminer l'impact de ces caractéristiques sur l'analyse de couverture structurelle selon les recommandations du sous-paragraphe 6.4.4.2.

b. Pour mettre en oeuvre certaines caractéristiques, les compilateurs de certains langages peuvent générer un code objet qui ne soit pas directement traçable vers le Code Source, par exemple, initialisation, mécanisme incorporé de détection des erreurs ou de traitement des exceptions (sous-paragraphe 6.4.4.2 b). La planification doit fournir un moyen de détecter ce code objet, d'assurer la couverture de vérification et d'en définir les moyens dans le plan approprié.

c. Si l'on introduit un nouveau compilateur, un nouvel éditeur de liens, ou une nouvelle version de chargeur, ou si l'on modifie les options du compilateur au cours du cycle de vie du logiciel, les tests et analyses de couverture antérieurs peuvent ne plus être valables. La planification de la vérification doit fournir un moyen de revérification qui soit cohérent avec les recommandations de la section 6 et du paragraphe 12.1.3.

4.4.3 Environnement de test du logiciel

La planification de l'environnement de test d'un logiciel a pour but de définir les méthodes, outils, procédures, et matériels à utiliser pour tester les produits du processus d'intégration. On peut procéder à ce test au moyen de la machine cible, d'un émulateur de la machine cible, ou d'un simulateur sur la machine hôte. Les directives sont les suivantes :

a. L'émulateur ou le simulateur doivent être qualifiés de la manière décrite dans la sous-section 12.2.

b. On doit tenir compte des différences entre la machine cible et l'émulateur ou le simulateur, et des effets de ces différences sur la capacité de détection d'erreurs et de vérification de la fonctionnalité. La détection de ces erreurs doit être assurée par d'autres activités du processus de vérification du logiciel et spécifiée dans le Plan de Vérification du Logiciel.

4.5 REGLES DE DEVELOPPEMENT DES LOGICIELS

Les règles de développement d'un logiciel ont pour but de définir les règles et les contraintes relatives à son développement. Les règles de développement d'un logiciel comprennent les Règles de Spécifications du Logiciel, les Règles de Conception du Logiciel, et les Règles de Codage du Logiciel. Le processus de vérification d'un logiciel utilise ces règles comme base d'évaluation de la conformité des produits effectifs d'un processus vis à vis des produits projetés. Les directives pour les règles de développement d'un logiciel sont les suivantes :

a. Les règles de développement d'un logiciel doivent être conformes à la section 11.

b. The software development standards should enable software components of a given software product or related set of products to be uniformly designed and implemented.

c. The software development standards should disallow the use of constructs or methods that produce outputs that cannot be verified or that are not compatible with safety-related requirements.

*NOTE: In developing standards, consideration can be given to previous experience. Constraints and rules on development, design and coding methods can be included to control complexity. Defensive programming practices may be considered to improve robustness.*

## 4.6 REVIEW AND ASSURANCE OF THE SOFTWARE PLANNING PROCESS

Reviews and assurance of the software planning process are conducted to ensure that the software plans and software development standards comply with the guidelines of this document and means are provided to execute them. Guidance includes:

a. The chosen methods will enable the objectives of this document to be satisfied.

b. The software life cycle processes can be applied consistently.

c. Each process produces evidence that its outputs can be traced to their activity and inputs, showing the degree of independence of the activity, the environment, and the methods to be used.

d. The outputs of the software planning process are consistent and comply with section 11.

b. Les règles de développement d'un logiciel doivent permettre la conception et la réalisation uniformes des composants d'un produit logiciel ou d'un ensemble associé de produits logiciels donnés.

c. Les règles de développement d'un logiciel doivent interdire l'utilisation de constructions ou de méthodes générant des sorties impossibles à vérifier ou incompatibles avec les exigences relatives à la sécurité.

*NOTA :* *Dans le développement des règles, on doit tenir compte de l'expérience acquise. On peut inclure des contraintes et règles de développement, des méthodes de conception et de codage pour limiter la complexité. On peut prendre en considération les techniques de programmation défensive* pour améliorer la robustesse.

## 4.6 REVUES ET ASSURANCE DE LA PLANIFICATION DU LOGICIEL

Les revues et l'assurance de la planification sont menées dans le but de s'assurer que les plans et les règles de développement sont conformes aux recommandations de ce document et que les moyens de les exécuter existent. Les directives sont les suivantes :

a. Les méthodes choisies permettront de satisfaire les objectifs de ce document.

b. Les processus du cycle de vie du logiciel peuvent s'appliquer de manière cohérente.

c. Chaque processus fournit la preuve de la traçabilité de ses données de sortie vers ses activités et ses données d'entrée en mettant en évidence le degré d'indépendance de l'activité, l'environnement, et les méthodes à employer.

d. Les sorties de la planification sont cohérentes et conformes à la section 11.

# SECTION 5

# SOFTWARE DEVELOPMENT PROCESSES

This section discusses the objectives and activities of the software development processes. The software development processes are applied as defined by the software planning process (section 4) and the Software Development Plan (subsection 11.2). Table A-2 of Annex A is a summary of the objectives and outputs of the software development processes by software level. The software development processes are:

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

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

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

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

## 5.1 SOFTWARE REQUIREMENTS PROCESS

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

### 5.1.1 Software Requirements Process Objectives

The objectives of the software requirements process are:

a. High-level requirements are developed.

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

# SECTION 5

# DEVELOPPEMENT DU LOGICIEL

Cette section traite des objectifs et des activités de développement du logiciel. L'application de ces processus se fait conformément au processus de planification du logiciel (section 4) et au Plan de Développement du Logiciel (sous-section 11.2). Le Tableau A-2 de l'Annexe A constitue un résumé des objectifs et des sorties des processus de développement du logiciel par niveau logiciel. Les processus de développement du logiciel sont les suivants :

- Spécifications du logiciel.
- Conception du logiciel.
- Codage du logiciel.
- Intégration.

Le développement d'un logiciel génère un ou plusieurs niveaux de spécifications du logiciel. Les exigences de haut niveau sont générées directement par analyse des spécifications du système et de son architecture. Habituellement, ces exigences de haut niveau sont détaillées davantage au cours du processus de conception donnant naissance à un ou plusieurs niveaux successifs d'exigences de bas niveau. Cependant, s'il y a génération directe du Code Source à partir des exigences de haut niveau, on considère alors ces exigences de haut niveau comme étant également des exigences de bas niveau, et les recommandations pour les exigences de bas niveau sont aussi applicables.

Le développement d'une architecture de logiciel implique que des décisions soient prises sur la structure du logiciel. Pendant la conception du logiciel, on définit son architecture et on développe ses exigences de bas niveau. Les exigences de bas niveau sont des spécifications du logiciel à partir desquelles on peut réaliser le Code Source sans autre information.

Chaque développement du logiciel peut engendrer des exigences dérivées. Les exigences dérivées sont des exigences qui ne sont pas directement traçables vers les exigences de haut niveau. Un exemple d'une telle exigence dérivée est constitué par le besoin de développer un logiciel de traitement des interruptions pour la machine cible choisie. Les exigences de haut niveau peuvent comprendre des exigences dérivées, et les exigences de bas niveau également. Les effets des exigences dérivées sur les exigences de sécurité sont déterminées par l'analyse de sécurité du système.

## 5.1 SPECIFICATIONS DU LOGICIEL

Le processus des spécifications du logiciel utilise les produits du cycle de vie du système pour développer les exigences de haut niveau du logiciel. Ces exigences de haut niveau comprennent les exigences fonctionnelles, de performances, d'interface, et celles liées à la sécurité.

### 5.1.1 Objectifs des spécifications du logiciel

Les objectifs des spécifications du logiciel sont les suivants :

a. Développement des exigences de haut niveau.

b. Communication des exigences de haut niveau dérivées à l'analyse de sécurité du système.

### 5.1.2 Software Requirements Process Activities

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

The primary output of this process is the Software Requirements Data (subsection 11.9).

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

a. The system functional and interface requirements that are allocated to software should be analyzed for ambiguities, inconsistencies and undefined conditions.

b. Inputs to the software requirements process detected as inadequate or incorrect should be reported as feedback to the input source processes for clarification or correction.

c. Each system requirement that is allocated to software should be specified in the high-level requirements.

d. High-level requirements that address system requirements allocated to software to preclude system hazards should be defined.

e. The high-level requirements should conform to the Software Requirements Standards, and be verifiable and consistent.

f. The high-level requirements should be stated in quantitative terms with tolerances where applicable.

g. The high-level requirements should not describe design or verification detail except for specified and justified design constraints.

h. Each system requirement allocated to software should be traceable to one or more software high-level requirements.

i. Each high-level requirement should be traceable to one or more system requirements, except for derived requirements.

j. Derived high-level requirements should be provided to the system safety assessment process.

## 5.2 SOFTWARE DESIGN PROCESS

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

### 5.2.1 Software Design Process Objectives

The objectives of the software design process are :

a. The software architecture and low-level requirements are developed from the high-level requirements.

b. Derived low-level requirements are provided to the system safety assessment process.

### 5.1.2 Activités de définition des spécifications du logiciel

Les données d'entrée nécessaires à la définition des spécifications du logiciel comprennent les spécifications du système, l'interface avec le matériel, et l'architecture du système (si elle n'est pas comprise dans les spécifications) provenant du cycle de vie du système, le Plan de Développement du Logiciel et les Règles de Spécification du Logiciel provenant de la planification du logiciel. Une fois que les critères de transition planifiés ont été satisfaits, on peut utiliser ces données d'entrée pour développer les exigences de haut niveau du logiciel.

Le produit principal de ce processus est constitué par les Spécifications du Logiciel (sous-section 11.9).

La définition des spécifications du logiciel est achevée quand ses objectifs et ceux des processus intégraux associés ont été satisfaits. Les directives à considérer lors de ce processus sont les suivantes :

a. On doit analyser les spécifications fonctionnelles du système et d'interface allouées au logiciel dans le but de détecter les ambiguïtés, les incohérences, et les conditions non définies.

b. Les données d'entrée du processus des spécifications du logiciel détectées comme inadéquates ou incorrectes doivent être communiquées en retour aux processus produisant ces données d'entrée, pour clarification ou correction.

c. On doit préciser dans les exigences de haut niveau toutes les spécifications du système allouées au logiciel.

d. On doit définir les exigences de haut niveau qui répondent aux spécifications du système, allouées au logiciel pour éviter des comportements dangereux du système.

e. Les exigences de haut niveau doivent être conformes aux Règles de Spécification du Logiciel, et être vérifiables et cohérentes.

f. Les exigences de haut niveau doivent être énoncées en termes quantitatifs, avec tolérances éventuelles.

g. Les exigences de haut niveau ne doivent pas décrire des détails de conception ou de vérification, sauf en cas de contraintes de conception précisées et justifiées.

h. Toutes les spécifications du système allouées au logiciel doivent être traçables vers une ou plusieurs exigences de haut niveau du logiciel.

i. Toutes les exigences de haut niveau doivent être traçables vers une ou plusieurs des spécifications du système, sauf pour les exigences dérivées.

j. Les exigences de haut niveau dérivées doivent être communiquées à l'analyse de sécurité du système.

## 5.2 CONCEPTION DU LOGICIEL

Les exigences de haut niveau du logiciel sont affinées au moyen d'une ou de plusieurs itérations dans la conception du logiciel, afin de développer l'architecture du logiciel et les exigences de bas niveau qui peuvent être utilisées pour réaliser le Code Source.

### 5.2.1 Objectifs de la conception du logiciel

Les objectifs de la conception d'un logiciel sont les suivants :

a. L'architecture du logiciel et les exigences de bas niveau sont développées à partir des exigences de haut niveau.

b. Les exigences de bas niveau dérivées sont communiquées à l'analyse de sécurité du système.

5.2.2 Software Design Process Activities

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

The primary output of the process is the Design Description (subsection 11.10) which includes the software architecture and the low-level requirements.

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

a. Low-level requirements and software architecture developed during the software design process should conform to the Software Design Standards and be traceable, verifiable and consistent.

b. Derived requirements should be defined and analyzed to ensure that the higher level requirements are not compromised.

c. Software design process activities could introduce possible modes of failure into the software or, conversely, preclude others. The use of partitioning or other architectural means in the software design may alter the software level assignment for some components of the software. In such cases, additional data should be defined as derived requirements and provided to the system safety assessment process.

d. Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks and cross-channel comparisons.

e. Responses to failure conditions should be consistent with the safety-related requirements.

f. Inadequate or incorrect inputs detected during the software design process should be provided to either the system life cycle process, the software requirements process, or the software planning process as feedback for clarification or correction.

*NOTE: The current state of software engineering does not permit a quantitative correlation between complexity and the attainment of safety objectives. While no objective guidelines can be provided, the software design process should avoid introducing complexity because as the complexity of software increases, it becomes more difficult to verify the design and to show that the safety objectives of the software are satisfied.*

5.2.3 Designing for User-Modifiable Software

Guidance follows concerning the development of software that is designed to be modifiable by its users. A modifiable component is that part of the software that is intended to be changed by the user and a non-modifiable component is that which is not intended to be changed by the user. User-modifiable software may vary in complexity. Examples include a single memory bit used to select one of two equipment options, a table of messages, or a memory area that can be programmed, compiled, and linked for aircraft maintenance functions. Software of any level can include a modifiable component.

Guidance for designing user-modifiable software includes:

5.2.2 Activités de conception du logiciel

Les données d'entrée de la conception d'un logiciel sont les Spécifications du Logiciel, le Plan de Développement du Logiciel, et les Règles de Conception du Logiciel. Une fois que les critères de transition planifiés ont été satisfaits, on utilise les exigences de haut niveau dans la conception pour développer l'architecture du logiciel et les exigences de bas niveau. Ceci peut impliquer un ou plusieurs niveau d'exigences.

Le principal produit du processus est constitué par la Description de Conception (sous-section 11.10) qui comprend l'architecture du logiciel et les exigences de bas niveau.

La conception d'un logiciel est achevée quand ses objectifs et ceux des processus intégraux associés ont été satisfaits. Les directives à considérer lors de ce processus sont les suivantes :

a. Les exigences de bas niveau et l'architecture du logiciel développées au cours de la conception du logiciel doivent être conformes aux Règles de Conception du Logiciel et être traçables, vérifiables, et cohérentes.

b. Les exigences dérivées doivent être définies et analysées afin de s'assurer que les exigences de haut niveau ne sont pas altérées.

c. Les activités de conception d'un logiciel peuvent introduire des conditions de panne possibles dans le logiciel ou, inversement, en éviter d'autres. L'utilisation du partitionnement ou d'autres moyens architecturaux dans la conception du logiciel peut modifier l'affectation du niveau logiciel pour certains de ses composants. Dans de tels cas, des données additionnelles sont définies comme exigences dérivées et transmises au processus d'analyse de sécurité du système.

d. Les flux de contrôle et les flux de données doivent être surveillés quand les exigences de sécurité l'imposent, par exemple, par des chiens de garde ou des comparaisons par contrôle de vraisemblance et comparaison entre voies.

e. Les réponses à des conditions de pannes doivent être cohérentes avec les exigences de sécurité.

f. Les données d'entrée inadéquates ou incorrectes, détectées au cours de la conception du logiciel, doivent être communiquées soit au cycle de vie du système, soit à la définition des spécifications du logiciel, soit à la planification du logiciel, dans un but de clarification ou de correction.

*NOTA : L'état actuel de la technique des logiciels ne permet pas une corrélation quantitative entre la complexité et la satisfaction des objectifs de sécurité. Comme il n'est pas possible de donner de recommandations objectives, la conception d'un logiciel doit éviter d'introduire de la complexité, car quand la complexité d'un logiciel augmente, il devient plus difficile d'en vérifier la conception et de démontrer que les objectifs de sécurité sont satisfaits.*

5.2.3 Conception d'un logiciel modifiable par l'utilisateur

Les directives ci-dessous concernent le développement des logiciels conçus pour être modifiables par les utilisateurs. Un composant modifiable est une partie de logiciel prévue pour être modifiée par l'utilisateur et un composant non modifiable est un composant qui n'est pas prévu pour être modifié par l'utilisateur. Les logiciels modifiables par l'utilisateur peuvent être de complexité variable. Un bit employé pour choisir entre deux options d'équipement, une table de messages, ou une zone mémoire réservée pour des fonctions de maintenance de l'aéronef pouvant accueillir un programme compilé et chargé par l'utilisateur sont des exemples d'un tel logiciel. Un logiciel d'un niveau quelconque peut inclure un composant modifiable.

Les directives pour la conception des logiciels modifiables par l'utilisateur sont les suivantes :

a. The non-modifiable component should be protected from the modifiable component to prevent interference in the safe operation of the non-modifiable component. This protection can be enforced by hardware, by software, by the tools used to make the change, or by a combination of the three.

b. The applicant-provided means should be shown to be the only means by which the modifiable component can be changed.

## 5.3 SOFTWARE CODING PROCESS

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

### 5.3.1 Software Coding Process Objectives

The objective of the software coding process is:

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

### 5.3.2 Software Coding Process Activities

The coding process inputs are the low-level requirements and software architecture from the software design process, and the Software Development Plan and the Software Code Standards. The software coding process may be entered or re-entered when the planned transition criteria are satisfied. The Source Code is produced by this process based upon the software architecture and the low-level requirements.

The primary results of this process are Source Code (subsection 11.11) and object code.

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

a. The Source Code should implement the low-level requirements and conform to the software architecture.

b. The Source Code should conform to the Software Code Standards.

c. The Source Code should be traceable to the Design Description.

d. Inadequate or incorrect inputs detected during the software coding process should be provided to the software requirements process, software design process or software planning process as feedback for clarification or correction.

## 5.4 INTEGRATION PROCESS

The target computer, and the Source Code and object code from the software coding process are used with the linking and loading data (subsection 11.16) in the integration process to develop the integrated airborne system or equipment.

### 5.4.1 Integration Process Objectives

The objective of the integration process is:

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

a. Le composant non modifiable doit être protégé du composant modifiable de manière à garantir un fonctionnement sûr du composant non modifiable. Cette protection peut être établie par le matériel, par le logiciel, par les outils employés pour exécuter les modifications, ou par la combinaison des trois.

b. On doit montrer que les moyens fournis par le postulant sont les seuls qui permettent la modification du composant modifiable.

## 5.3 CODAGE DU LOGICIEL

Dans le codage d'un logiciel, le Code Source est réalisé à partir de l'architecture du logiciel et de ses exigences de bas niveau.

### 5.3.1 Objectifs du codage d'un logiciel

L'objectif du codage d'un logiciel est le suivant :

a. Un Code Source est développé ; il est traçable, vérifiable, cohérent, et il réalise correctement les exigences de bas niveau.

### 5.3.2 Activités du codage d'un logiciel

Les données d'entrée du codage d'un logiciel sont les exigences de bas niveau et l'architecture du logiciel issues du processus de conception, le Plan de Développement du Logiciel et les Règles de Codage du Logiciel. On peut engager ou réengager le codage du logiciel lorsque les critères de transition planifiés sont satisfaits. Le Code Source est généré par ce processus à partir de l'architecture du logiciel et de ses exigences de bas niveau.

Les principaux produits de ce processus sont le Code Source (sous-section 11.11) et le code objet.

Le codage d'un logiciel est achevé quand ses objectifs et ceux des processus intégraux associés sont satisfaits. Les directives à considérer lors de ce processus sont les suivantes :

a. Le Code Source doit réaliser les exigences de bas niveau et être conforme à l'architecture du logiciel.

b. Le Code Source doit être conforme aux Règles de Codage du Logiciel.

c. Le Code Source doit être traçable vers la Description de Conception.

d. Les données d'entrée inadéquates ou incorrectes détectées au cours du codage du logiciel doivent être communiqués en retour à la définition des spécifications du logiciel, à sa conception, ou à sa planification pour clarification ou correction.

## 5.4 INTEGRATION

On utilise dans l'intégration la machine cible, le Code Source et le code objet issus du codage du logiciel, ainsi que les résultats d'édition des liens et de chargement (sous-section 11.16), afin de développer le système ou équipement de bord intégré.

### 5.4.1 Objectifs de l'intégration

L'objectif de l'intégration est le suivant :

a. Chargement du Code Objet Exécutable dans la machine cible pour l'intégration matériel/logiciel.

5.4.2 Integration Process Activities

The integration process consists of software integration and hardware/software integration.

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

The outputs of the integration process are the Executable Object Code, as defined in subsection 11.12, and the linking and loading data. The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Guidance for this process includes:

a. The Executable Object Code should be generated from the Source Code and linking and loading data.

b. The software should be loaded into the target computer for hardware/software integration.

c. Inadequate or incorrect inputs detected during the integration process should be provided to the software requirements process, the software design process, the software coding process or the software planning process as feedback for clarification or correction.

5.4.3 Integration Considerations

The following are considerations for deactivated code and software patches. An airborne system or equipment may be designed to include several configurations, not all of which are intended to be used in every application. This can lead to deactivated code that cannot be executed or data that is not used. This differs from dead code which is defined in the glossary and discussed in subparagraph 6.4.4.3. Guidance for deactivated code and patches includes:

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

b. The methods for handling deactivated code should comply with the software plans.

c. Patches should not be used in software submitted for use in a certified aircraft or engine to implement changes in requirements or architecture, or changes found necessary as a result of software verification process activities. Patches may be used on a limited, case-by-case basis, for example, to resolve known deficiencies in the software development environment, such as a known compiler problem.

d. When a patch is used, these should be available:

   (1) Confirmation that the software configuration management process can effectively track the patch.

   (2) Regression analysis to provide evidence that the patch satisfies all objectives of the software developed by normal methods.

   (3) Justification in the Software Accomplishment Summary for the use of a patch.

5.4.2 Activités de l'intégration

L'intégration se compose de l'intégration du logiciel et de l'intégration matériel/logiciel.

On peut engager ou réengager le processus d'intégration une fois que les critères de transition planifiés ont été satisfaits. Les données d'entrée de l'intégration sont constituées par l'architecture du logiciel issues de sa conception, et par le Code Source et le code objet issus du codage du logiciel.

Les produits de l'intégration sont constitués par le Code Objet Exécutable, de la manière définie dans la sous-section 11.12, et par les données d'édition de liens et de chargement. L'intégration est achevée quand ses objectifs et ceux des processus intégraux associés sont satisfaits. Les directives à considérer lors de ce processus sont les suivantes :

a. Le Code Objet Exécutable doit être généré à partir du Code Source et des données d'édition de liens et de chargement.

b. Le logiciel doit être chargé dans la machine cible pour l'intégration matériel/logiciel.

c. Les données d'entrée inadéquates ou incorrectes détectées au cours de l'intégration doivent être communiquées en retour à la définition des spécifications du logiciel, à sa conception, à son codage, ou à sa planification, pour clarification ou correction.

5.4.3 Considérations relatives à l'intégration

Les considérations ci-dessous sont relatives aux codes désactivés et aux "patches". On peut concevoir un système ou équipement de bord de manière à y inclure plusieurs configurations, toutes n'étant pas prévues pour être employées dans chaque application. Ceci peut conduire à des codes désactivés inexécutables ou à des données non utilisées. Ceci est différent du code mort défini dans le glossaire et traité dans le sous-paragraphe 6.4.4.3. Les directives pour les codes désactivés et les "patches" incluent :

a. La preuve doit être fournie que le code désactivé ne peut pas être activé pour les environnements où son utilisation n'est pas prévue. L'activation non prévue d'un code désactivé, due à des conditions anormales de fonctionnement du système, est de même nature que l'activation non prévue d'un code activable.

b. Les méthodes de traitement des codes désactivés doivent être conformes aux plans.

c. On ne doit pas utiliser dans les logiciels destinés à une utilisation sur un aéronef ou un moteur déjà certifié, des "patches" ayant pour but de réaliser des modifications d'exigences ou d'architecture, ou des modifications trouvées nécessaires suite à des activités de vérification de logiciel. On peut utiliser les "patches" de manière limitée, au cas par cas, pour résoudre des insuffisances connues dans l'environnement de développement d'un logiciel, telles qu'un problème connu de compilateur.

d. Dans le cas d'utilisation d'un "patche", on doit disposer des éléments suivants :

   (1) Confirmation du fait que la gestion de configuration du logiciel peut suivre efficacement le "patche".

   (2) Analyse de régression pour démontrer que le "patche" satisfait tous les objectifs du logiciel développé selon les méthodes normales.

   (3) Justification de l'utilisation d'un "patche" dans le Résumé des Travaux Réalisés.

## 5.5 TRACEABILITY

Traceability guidance includes:

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

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

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

## 5.5 TRAÇABILITE

Les directives de traçabilité sont les suivantes :

a. La traçabilité entre les spécifications du système et celles du logiciel doit être fournie pour vérifier la réalisation complète des spécifications du système et donner une visibilité des exigences dérivées.

b. La traçabilité entre les exigences de bas niveau et les exigences de haut niveau doit être fournie pour vérifier les exigences dérivées et les décisions sur l'architecture prises au cours de la conception du logiciel, et permettre la vérification de la réalisation complète des exigences de haut niveau.

c. La traçabilité entre le Code Source et les exigences de bas niveau doit être fournie pour vérifier l'absence de Code Source non documenté et la vérification de la réalisation complète des exigences de bas niveau.

# SECTION 6

# SOFTWARE VERIFICATION PROCESS

This section discusses the objectives and activities of the software verification process. Verification is a technical assessment of the results of both the software development processes and the software verification process. The software verification process is applied as defined by the software planning process (section 4) and the Software Verification Plan (subsection 11.3).

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

Tables A-3 through A-7 of Annex A contain a summary of the objectives and outputs of the software verification process, by software level.

NOTE: *For lower software levels, less emphasis is on:*

- *Verification of low-level requirements.*
- *Verification of the software architecture.*
- *Degree of test coverage.*
- *Control of verification procedures.*
- *Independence of software verification process activities.*
- *Overlapping software verification process activities, that is, multiple verification activities, each of which may be capable of detecting the same class of error.*
- *Robustness testing.*
- *Verification activities with an indirect effect on error prevention or detection, for example, conformance to software development standards.*

## 6.1 SOFTWARE VERIFICATION PROCESS OBJECTIVES

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

a. The system requirements allocated to software have been developed into software high-level requirements that satisfy those system requirements.

b. The high-level requirements have been developed into software architecture and low-level requirements that satisfy the high-level requirements. If one or more levels of software requirements are developed between high-level requirements and low-level requirements, the successive levels of requirements are developed such that each successively lower level satisfies its higher level requirements. If code is generated directly from high-level requirements, this objective does not apply.

c. The software architecture and low-level requirements have been developed into Source Code that satisfies the low-level requirements and software architecture.

d. The Executable Object Code satisfies the software requirements.

e. The means used to satisfy these objectives are technically correct and complete for the software level.

# SECTION 6

# VERIFICATION DU LOGICIEL

Cette section traite des objectifs et des activités du processus de vérification du logiciel. La vérification est une évaluation technique à la fois des produits du développement du logiciel et de ceux de son processus de vérification. Le processus de vérification du logiciel s'applique de la manière définie par le processus de planification (section 4) et par le Plan de Vérification du Logiciel (sous-section 11.3).

La vérification n'est pas simplement du test. Le test ne peut pas, en général, démontrer l'absence d'erreurs. Il en résulte que les sous-sections suivantes utiliseront le terme "vérifier" à la place du terme "tester" chaque fois que les objectifs du processus de vérification du logiciel seront constitués d'une combinaison de revues, d'analyses, et de tests.

Les Tableaux A-3 à A-7 de l'annexe A résument les objectifs et les sorties du processus de vérification du logiciel, par niveau logiciel.

*NOTA : Pour les niveaux logiciels les plus bas, on insiste moins sur :*

- *La vérification des exigences de bas niveau.*
- *La vérification de l'architecture du logiciel.*
- *Le degré de couverture des tests.*
- *Le contrôle des procédures de vérification.*
- *L'indépendance des activités du processus de vérification du logiciel.*
- *Le recouvrement des activités du processus de vérification du logiciel, c'est-à-dire les activités multiples de vérification, chacune d'entre elles étant capable de détecter la même classe d'erreurs.*
- *Les tests de robustesse.*
- *Les activités de vérification ayant un effet indirect sur la prévention ou la détection des erreurs, par exemple la conformité avec les règles de développement du logiciel.*

## 6.1 OBJECTIFS DU PROCESSUS DE VERIFICATION DU LOGICIEL

Le processus de vérification du logiciel a pour but de détecter et de rendre compte des erreurs qui peuvent avoir été introduites au cours du développement du logiciel. L'élimination de ces erreurs est une des activités du développement du logiciel. Les objectifs généraux du processus de vérification du logiciel consistent à vérifier que :

a. Les spécifications du système allouées au logiciel ont été développées sous la forme d'exigences de haut niveau satisfaisant ces spécifications du système.

b. Les exigences de haut niveau ont été développées sous la forme d'une architecture du logiciel et d'exigences de bas niveau satisfaisant les exigences de haut niveau. Si plusieurs niveaux d'exigences sont nécessaires entre les exigences de haut niveau et celles de bas niveau, les niveaux successifs d'exigences sont développés de telle façon que chaque niveau inférieur satisfait les exigences des niveaux supérieurs. Si le code est généré directement à partir des exigences de haut niveau, le présent objectif n'est pas applicable.

c. L'architecture du logiciel et les exigences de bas niveau ont été développées sous la forme d'un Code Source satisfaisant les exigences de bas niveau et l'architecture du logiciel.

d. Le Code Objet Exécutable satisfait les spécifications du logiciel.

e. Les moyens utilisés pour satisfaire ces objectifs sont techniquement corrects et complets pour le niveau logiciel.

## 6.2 SOFTWARE VERIFICATION PROCESS ACTIVITIES

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

The outputs of the software verification process are recorded in Software Verification Cases and Procedures (subsection 11.13) and Software Verification Results (subsection 11.14).

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

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

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

Guidance for the software verification activities includes:

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

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

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

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

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

## 6.3 SOFTWARE REVIEWS AND ANALYSES

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

## 6.2 ACTIVITES DU PROCESSUS DE VERIFICATION DU LOGICIEL

Les objectifs du processus de vérification d'un logiciel sont satisfaits au moyen d'une combinaison de revues et analyses, du développement de jeux et procédures de test et de l'exécution nécessaire de ces procédures. Les revues et analyses donnent une évaluation de la précision, de l'exhaustivité, et de la capacité des spécifications du logiciel, de son architecture et du Code Source à être vérifiées. Le développement de jeux de test peut fournir une évaluation complémentaire de la cohérence interne et de l'exhaustivité des exigences. L'exécution des procédures de test fournit une démonstration de la conformité avec les exigences.

Les entrées du processus de vérification du logiciel comprennent les spécifications du système, les spécifications et l'architecture du logiciel, les données de traçabilité, le Code Source, le Code Objet Exécutable, et le Plan de Vérification du Logiciel.

Les produits du processus de vérification d'un logiciel sont enregistrés dans les Jeux et Procédures de Vérification du Logiciel (sous-section 11.13) et les Résultats de Vérification du Logiciel (sous-section 11.14).

La nécessité de pouvoir vérifier les spécifications après leur mise en application dans le logiciel peut imposer des exigences ou contraintes complémentaires aux processus de développement du logiciel.

Le processus de vérification fournit la traçabilité entre la mise en application des spécifications du logiciel et leur vérification.

- La traçabilité entre les spécifications du logiciel et les jeux de test est déterminée par une analyse de couverture fondée sur les exigences.
- La traçabilité entre la structure du code et les jeux de test est déterminée par une analyse de couverture structurelle.

Les directives pour les activités de vérification du logiciel sont les suivantes :

a. Les exigences de haut niveau doivent être vérifiées ainsi que la traçabilité vers ces exigences.

b. Les résultats de l'analyse de traçabilité ainsi que des analyses de couverture fondée sur les exigences et de couverture structurelle doivent démontrer que chaque spécification du logiciel est traçable vers le code qui la met en application ainsi que vers la revue, l'analyse, ou le jeux de test qui la vérifie.

c. Si le code testé n'est pas identique au logiciel embarqué, on doit préciser et justifier ces différences.

d. S'il n'est pas possible de vérifier certaines spécifications particulières du logiciel en faisant fonctionner ce dernier dans un environnement réaliste de test, on doit utiliser d'autres moyens et justifier leur utilisation pour satisfaire les objectifs du processus de vérification du logiciel, définis dans le Plan de Vérification du Logiciel ou dans les Résultats de Vérification du Logiciel.

e. Les insuffisances et les erreurs découvertes au cours du processus de vérification du logiciel doivent être transmises au processus de développement du logiciel aux fins de clarification et de correction.

## 6.3 REVUES ET ANALYSES DU LOGICIEL

Les revues et analyses concernent les produits du développement du logiciel et son processus de vérification. Les revues et les analyses diffèrent en ce que les analyses fournissent une preuve reproductible de l'exactitude, alors que les revues donnent une évaluation qualitative de cette exactitude. Une revue peut être constituée par une inspection du produit d'un processus s'appuyant sur une liste de contrôle (check list) ou une aide analogue. Une analyse peut examiner en détail la fonctionnalité, les performances, la traçabilité et les implications relatives à la sécurité d'un composant logiciel, ainsi que ses rapports avec d'autres composants du système ou équipement de bord.

6.3.1 Reviews and Analyses of the High-Level Requirements

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

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

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

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

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

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

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

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

6.3.2 Reviews and Analyses of the Low-Level Requirements

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

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

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

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

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

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

6.3.1 Revues et analyses des exigences de haut niveau

Ces revues et analyses ont pour objectif de détecter et de rendre compte des erreurs d'exigences qui peuvent avoir été introduites au cours de la définition des spécifications du logiciel. Ces revues et analyses confirment que les exigences de haut niveau satisfont les objectifs suivants :

a. Conformité avec les spécifications du système : L'objectif est de s'assurer que les fonctions du système à réaliser par le logiciel sont définies, que les spécifications fonctionnelles, de performances et relatives à la sécurité du système sont satisfaites par les exigences de haut niveau du logiciel, et que les exigences dérivées et la raison de leur existence sont correctement définies.

b. Précision et cohérence : L'objectif est de s'assurer que chaque exigence de haut niveau est précise, sans ambiguïté, et suffisamment détaillée, et que les exigences ne sont pas mutuellement conflictuelles.

c. Compatibilité avec la machine cible : L'objectif est de s'assurer qu'il n'existe pas de conflits entre les exigences de haut niveau et les caractéristiques matérielles et logicielles de la machine cible, en particulier en ce qui concerne les temps de réponse du système et les dispositifs d'entrée et de sortie.

d. Capacité à être vérifié : L'objectif est de s'assurer que toutes les exigences de haut niveau sont vérifiables.

e. Conformité avec les règles : L'objectif est de s'assurer que les Règles de Spécification du Logiciel ont été respectées au cours de la définition des exigences et que les écarts avec les règles sont justifiés.

f. Traçabilité : L'objectif est de s'assurer que les spécifications fonctionnelles et de performances, ainsi que celles relatives à la sécurité et allouées au logiciel ont été développées sous la forme d'exigences de haut niveau du logiciel.

g. Aspects relatifs aux algorithmes : L'objectif est de s'assurer de la précision et du comportement des algorithmes proposés, en particulier dans les zones de discontinuité.

6.3.2 Revues et analyses d'exigences de bas niveau

Ces revues et analyses ont pour objectif de détecter et de rendre compte des erreurs d'exigences qui ont pu être introduites au cours de la conception du logiciel. Ces revues et analyses confirment que les exigences de bas niveau satisfont les objectifs suivants :

a. Conformité avec les exigences de haut niveau : L'objectif est de s'assurer que les exigences de bas niveau du logiciel satisfont ses exigences de haut niveau et que les exigences dérivées et les bases de conception justifiant leur existence sont correctement définies.

b. Précision et cohérence : L'objectif est de s'assurer que chaque exigence de bas niveau est précise et sans ambiguïté et que les exigences ne sont pas mutuellement conflictuelles.

c. Compatibilité avec la machine cible : L'objectif est de s'assurer qu'il n'existe pas de conflits entre les exigences de bas niveau et les caractéristiques matérielles et logicielles de la machine cible, en particulier en ce qui concerne l'utilisation de ressources (telles qu'un chargement par bus), les temps de réponse du système, et les dispositifs d'entrée et de sortie.

d. Capacité à être vérifié : L'objectif est de s'assurer que toutes les exigences de bas niveau sont vérifiables.

e. Conformité avec les règles : L'objectif est de s'assurer que les Règles de Conception du Logiciel ont été respectées au cours de la conception du logiciel et, que les écarts avec les règles sont justifiés.

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

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

### 6.3.3 Reviews and Analyses of the Software Architecture

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

a. Compatibility with the high-level requirements: The objective is to ensure that the software architecture does not conflict with the high-level requirements, especially functions that ensure system integrity, for example, partitioning schemes.

b. Consistency: The objective is to ensure that a correct relationship exists between the components of the software architecture. This relationship exists via data flow and control flow.

c. Compatibility with the target computer: The objective is to ensure that no conflicts exist, especially initialization, asynchronous operation, synchronization and interrupts, between the software architecture and the hardware/software features of the target computer.

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

e. Conformance to standards: The objective is to ensure that the Software Design Standards were followed during the software design process and that deviations to the standards are justified, especially complexity restrictions and design constructs that would not comply with the system safety objectives.

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

### 6.3.4 Reviews and Analyses of the Source Code

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

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

b. Compliance with the software architecture: The objective is to ensure that the Source Code matches the data flow and control flow defined in the software architecture.

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

f. Traçabilité : L'objectif est de s'assurer que les exigences de haut niveau et les exigences dérivées ont été développées sous la forme d'exigences de bas niveau.

g. Aspects relatifs aux algorithmes : L'objectif est de s'assurer de la précision et du comportement des algorithmes proposés, en particulier dans les zones de discontinuité.

### 6.3.3 Revues et analyses de l'architecture du logiciel

Ces revues et analyses ont pour objectif de détecter et de rendre compte des erreurs qui ont pu être introduites au cours du développement de l'architecture du logiciel. Ces revues et analyses confirment que l'architecture du logiciel satisfait les objectifs suivants :

a. Compatibilité avec les exigences de haut niveau : L'objectif est de s'assurer que l'architecture du logiciel n'entre pas en conflit avec les exigences de haut niveau, en particulier les fonctions assurant l'intégrité du système, par exemple les schémas de partitionnement.

b. Cohérence : L'objectif est de s'assurer qu'il existe une relation correcte entre les composants de l'architecture du logiciel. Cette relation existe par les flux de données et les flux de contrôle.

c. Compatibilité avec la machine cible : L'objectif est de s'assurer qu'il n'existe pas de conflits, en particulier d'initialisation, de fonctionnement asynchrone, de synchronisation et d'interruptions, entre l'architecture du logiciel et les caractéristiques matérielles et logicielles de la machine cible.

d. Capacité à être vérifié : L'objectif est de s'assurer que l'architecture du logiciel est vérifiable, par exemple qu'il n'y a pas d'algorithmes récursifs non bornés.

e. Conformité avec les règles : L'objectif est de s'assurer que les Règles de Conception du Logiciel ont été respectées au cours de la conception du logiciel et que les écarts avec les règles sont justifiés, en particulier les restrictions de complexité et les constructions de conception qui ne seraient pas conformes aux objectifs de sécurité du système.

f. Intégrité du partitionnement : L'objectif est de s'assurer que les violations du partitionnement sont évitées.

### 6.3.4 Revues et analyses du Code Source

L'objectif est de détecter et de rendre compte des erreurs qui ont pu être introduites au cours du codage du logiciel. Ces revues et analyses confirment que les produits du codage du logiciel sont précis, complets, et vérifiables. Les questions principales comprennent l'exactitude du code par rapport aux spécifications du logiciel et à son architecture, et la conformité vis à vis des Règles de Codage du Logiciel. Ces revues et analyses sont habituellement limitées au Code Source. Les points à vérifier sont les suivants :

a. Conformité avec les exigences de bas niveau : L'objectif est de s'assurer que le Code Source est précis et complet vis à vis des exigences de bas niveau du logiciel, et qu'aucun Code Source ne met en application une fonction non documentée.

b. Conformité avec l'architecture du logiciel : l'objectif est de s'assurer que le Code Source est adapté au flux de données et au flux de contrôle définis dans l'architecture du logiciel.

c. Capacité à être vérifié : L'objectif est de s'assurer que le Code Source ne renferme pas d'instructions ni de structures invérifiables et qu'il n'est pas nécessaire de l'altérer pour le tester.

d. Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, especially complexity restrictions and code constraints that would be consistent with the system safety objectives. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and the complexity of logical or numeric expressions. This analysis also ensures that deviations to the standards are justified.

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

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

### 6.3.5 Reviews and Analyses of the Outputs of the Integration Process

The objective is to ensure that the results of the integration process are complete and correct. This could be performed by a detailed examination of the linking and loading data and memory map. The topics should include:

a. Incorrect hardware addresses.

b. Memory overlaps.

c. Missing software components.

### 6.3.6 Reviews and Analyses of the Test Cases, Procedures and Results

The objective of these reviews and analyses is to ensure that the testing of the code was developed and performed accurately and completely. The topics should include:

a. Test cases: The verification of test cases is presented in paragraph 6.4.4.

b. Test procedures: The objective is to verify that the test cases were accurately developed into test procedures and expected results.

c. Test results: The objective is to ensure that the test results are correct and that discrepancies between actual and expected results are explained.

## 6.4 SOFTWARE TESTING

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

Figure 6-1 is a diagram of the software testing activities. The objectives of the three types of testing in the figure are:

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

d. Conformité avec les règles : L'objectif est de s'assurer que les Règles de Codage du Logiciel ont été respectées au cours du développement du code, en particulier les restrictions de complexité et les contraintes de code destinées à les rendre conformes aux objectifs de sécurité du système. La complexité comprend le degré de couplage entre les composants logiciels, les niveaux d'imbrication des structures de contrôle, et la complexité des expressions logiques ou numériques. L'analyse s'assure également que les écarts avec les règles sont justifiés.

e. Traçabilité : L'objectif est de s'assurer que les exigences de bas niveau du logiciel ont été développées dans le Code Source.

f. Précision et cohérence : L'objectif est d'établir l'exactitude et la cohérence du Code Source, et notamment pour l'utilisation des piles, les débordements et la résolution de l'arithmétique en virgule fixe, les conflits de ressources, les temps d'exécution les plus défavorables, la gestion des interruptions, les variables ou constantes inutilisées et l'altération des données due à des conflits de tâches ou d'interruptions..

### 6.3.5 Revues et analyses des sorties du processus d'intégration

L'objectif est de s'assurer que les résultats du processus d'intégration sont complets et corrects. On peut y parvenir au moyen d'un examen détaillé des données d'édition de liens et de chargement et de la configuration mémoire. Les points à vérifier comprennent :

a. Les adresses matérielles incorrectes.

b. Les recouvrements de mémoire.

c. Les composants logiciels manquants.

### 6.3.6 Revues et analyses des jeux, procédures, et résultats de test

Ces revues et analyses ont pour objectif de s'assurer que le test du code a été réalisé de manière précise et complète. Les points à vérifier comprennent :

a. Jeux de test : La vérification des jeux de test est présentée dans le paragraphe 6.4.4.

b. Procédures de test : L'objectif est de vérifier que les jeux de test ont été développés avec précision sous la forme de procédures de test et de résultats attendus.

c. Résultats de test : L'objectif est de s'assurer que les résultats de test sont corrects et que les écarts entre les résultats réels et les résultats attendus sont expliqués.

## 6.4 TESTS DU LOGICIEL

Les tests d'un logiciel embarqué ont deux objectifs complémentaires. L'un est de démontrer que le logiciel satisfait ses spécifications. Le second objectif est de démontrer avec un degré élevé de confiance qu'on a éliminé les erreurs qui pourraient mener à des conditions de panne inacceptables telles qu'elles sont déterminées par l'analyse de sécurité du système.

La Figure 6-1 représente un diagramme des activités de test d'un logiciel. Les objectifs des trois types de tests de la figure sont les suivants :

- Test d'intégration matériel/logiciel : Vérification du fonctionnement correct du logiciel dans l'environnement de la machine cible.

- Test d'intégration du logiciel : Vérification des relations mutuelles entre les exigences et les composants logiciels et vérification de la mise en application des spécifications du logiciel et de ses composants dans son architecture.

- Low-level testing: To verify the implementation of software low-level requirements.

  *NOTE: If a test case and its corresponding test procedure are developed and executed for hardware/software integration testing or software integration testing and satisfy the requirements-based coverage and structural coverage, it is not necessary to duplicate the test for low-level testing. Substituting nominally equivalent low-level tests for high-level tests may be less effective due to the reduced amount of overall functionality tested.*

To satisfy the software testing objectives:

a. Test cases should be based primarily on the software requirements.

b. Test cases should be developed to verify correct functionality and to establish conditions that reveal potential errors.

c. Software requirements coverage analysis should determine what software requirements were not tested.

d. Structural coverage analysis should determine what software structures were not exercised.

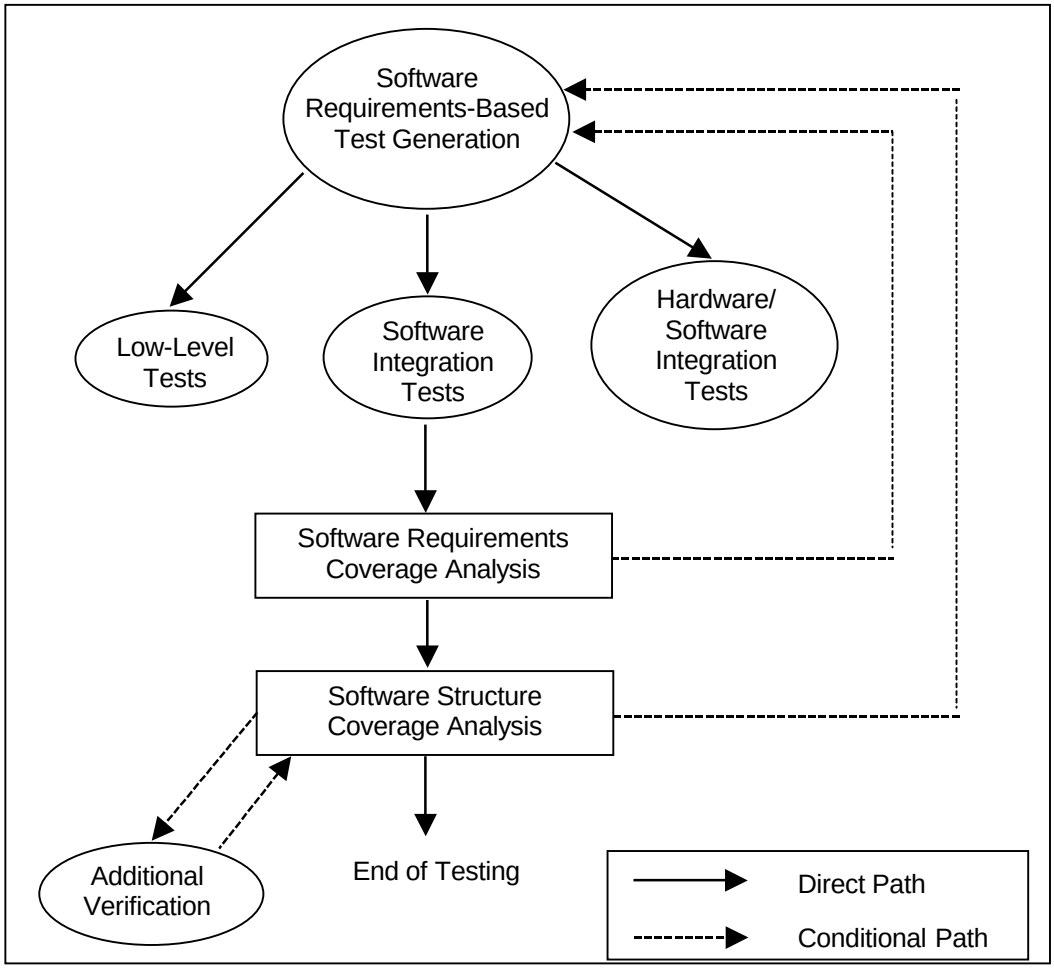



FIGURE 6-1: SOFTWARE TESTING ACTIVITIES

- Test de bas niveau : Vérification des exigences de bas niveau du logiciel.

  *NOTA :* *Si un jeu de test et les procédures de test correspondantes sont développés et exécutés pour un test d'intégration matériel/logiciel ou un test d'intégration de logiciel, et s'ils satisfont la couverture fondée sur les exigences et la couverture structurelle, il n'est pas nécessaire de recommencer le test pour le test de bas niveau. La substitution de tests de bas niveau nominalement équivalents à des tests de haut niveau peut être moins efficace du fait de la moindre quantité de fonctionnalité globale essayée.*

Pour satisfaire les objectifs de test d'un logiciel :

a. Les jeux de test doivent être principalement élaborés à partir des spécifications du logiciel.

b. Les jeux de test doivent être développés de manière à vérifier l'exactitude de la fonctionnalité et à établir des conditions qui révèlent les erreurs potentielles.

c. L'analyse de couverture des spécifications du logiciel doit déterminer lesquelles de ces spécifications n'ont pas été testées.

d. L'analyse de couverture structurelle doit déterminer celles des structures du logiciel qui n'ont pas été parcourues.

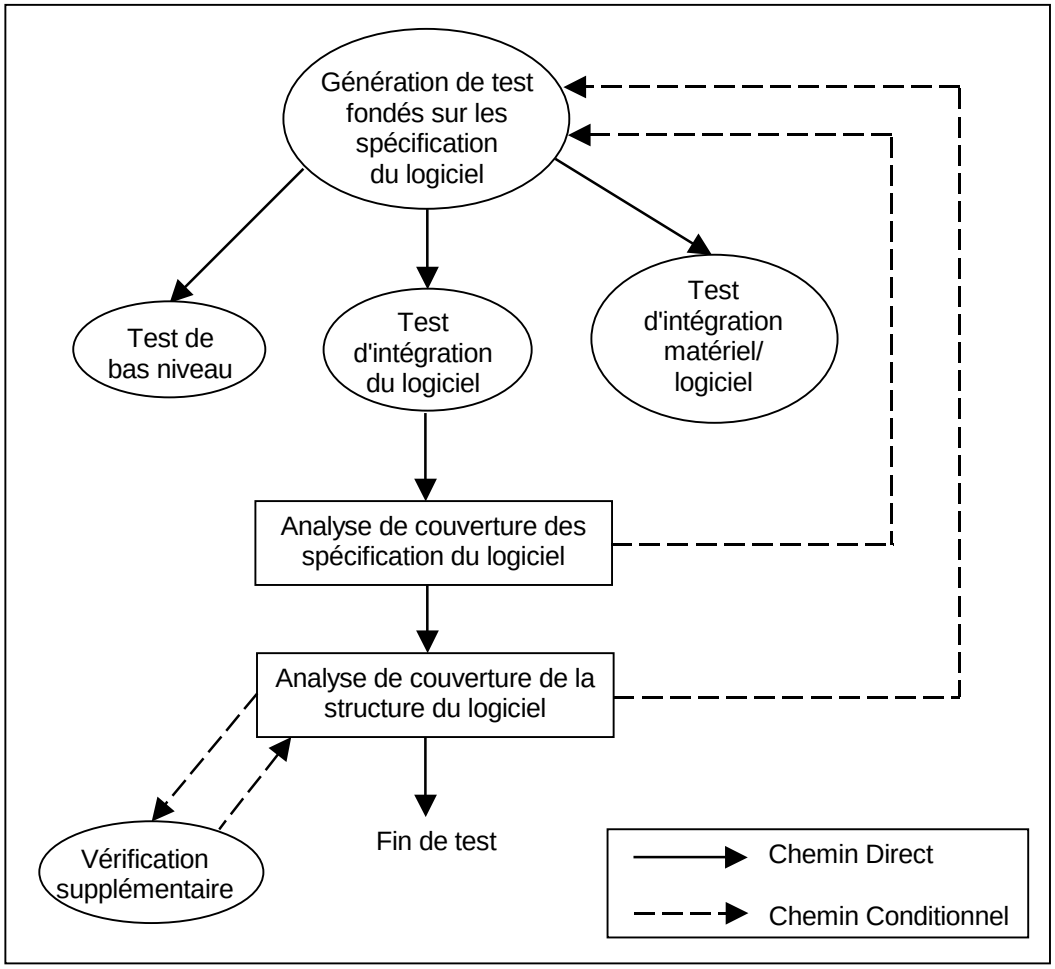



FIGURE 6-1 : ACTIVITES DE TEST DU LOGICIEL

6.4.1 Test Environment

More than one test environment may be needed to satisfy the objectives for software testing. An excellent test environment includes the software loaded into the target computer and tested in a high fidelity simulation of the target computer environment.

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

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

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

6.4.2 Requirements-Based Test Case Selection

Requirements-based testing is emphasized because this strategy has been found to be the most effective at revealing errors. Guidance for requirements-based test case selection includes:

a. To implement the software testing objectives, two categories of test cases should be included: normal range test cases and robustness (abnormal range) test cases.

b. The specific test cases should be developed from the software requirements and the error sources inherent in the software development processes.

6.4.2.1 Normal Range Test Cases

The objective of normal range test cases is to demonstrate the ability of the software to respond to normal inputs and conditions. Normal range test cases include:

a. Real and integer input variables should be exercised using valid equivalence classes and boundary values.

b. For time-related functions, such as filters, integrators and delays, multiple iterations of the code should be performed to check the characteristics of the function in context.

c. For state transitions, test cases should be developed to exercise the transitions possible during normal operation.

d. For software requirements expressed by logic equations, the normal range test cases should verify the variable usage and the Boolean operators.

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

6.4.1 Environnement de test

On peut avoir besoin de plus d'un environnement de test pour satisfaire les objectifs de test d'un logiciel. Un environnement de test excellent comprend le logiciel chargé dans la machine cible et essayé dans une simulation très fidèle de l'environnement de ce calculateur.

*NOTA : Dans de nombreux cas, la couverture des exigences et la couverture structurelle ne peuvent être obtenues que par un contrôle et une surveillance des entrées de test et de l'exécution plus précis que ceux généralement obtenus dans un environnement intégré. Il peut être nécessaire d'effectuer un tel test sur un petit composant logiciel isolé fonctionnellement des autres composants.*

Un crédit de certification peut être accordé à un test réalisé au moyen d'un émulateur de la machine cible ou d'un simulateur de la machine hôte. La recommandation concernant l'environnement de test est la suivante :

a. Les tests choisis doivent être réalisés dans l'environnement intégré de la machine cible, dans la mesure où certaines erreurs ne sont détectables que dans cet environnement.

6.4.2 Choix de jeux de test fondés sur les exigences

On insiste sur les tests fondés sur les exigences car il est apparu que cette stratégie était la plus efficace pour révéler les erreurs. Les directives pour le choix des jeux de test fondés sur les exigences sont :

a. Les objectifs de test du logiciel doivent être atteints au moyen de deux catégories de test : des jeux de test dans les plages de variation prévues et des jeux de test de robustesse (hors des plages de variation).

b. Les jeux de test spécifiques doivent être développés à partir des spécifications du logiciel et des sources d'erreurs inhérentes au développement de ce logiciel.

6.4.2.1 Jeux de test avec des données dans les plages de variation prévues

Les jeux de test avec des données dans les plages de variation prévues ont pour objectif de démontrer la capacité du logiciel à répondre à des données d'entrée et des conditions normales. Ces jeux de test contiennent notamment :

a. Des variables réelles et entières doivent être mises en oeuvre par le biais de classes d'équivalence valides et de valeurs limites.

b. Pour les fonctions liées au temps, tels que filtres, intégrateurs, et retards, on doit réaliser des itérations multiples afin de vérifier les caractéristiques de la fonction dans son contexte.

c. Pour les transitions d'états, on doit développer des jeux de test afin de mettre en oeuvre les transitions possibles en fonctionnement normal.

d. Pour les spécifications de logiciel exprimées par des équations logiques, les jeux de test doivent vérifier l'utilisation des variables et les opérateurs booléens.

*NOTA : Une méthode consiste à essayer toutes les combinaisons des variables. Pour les expressions complexes, cette méthode est peu pratique à cause du grand nombre de jeux de test nécessaires. On pourra développer une stratégie différente assurant la couverture requise. Par exemple, pour le Niveau A, on pourra vérifier les opérateurs booléens par analyse ou revue, et, compléter cette activité par des jeux de test assurant la couverture des conditions/décisions modifiée*.

6.4.2.2 Robustness Test Cases

The objective of robustness test cases is to demonstrate the ability of the software to respond to abnormal inputs and conditions. Robustness test cases include:

a. Real and integer variables should be exercised using equivalence class selection of invalid values.

b. System initialization should be exercised during abnormal conditions.

c. The possible failure modes of the incoming data should be determined, especially complex, digital data strings from an external system.

d. For loops where the loop count is a computed value, test cases should be developed to attempt to compute out-of-range loop count values, and thus demonstrate the robustness of the loop-related code.

e. A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly.

f. For time-related functions, such as filters, integrators and delays, test cases should be developed for arithmetic overflow protection mechanisms.

g. For state transitions, test cases should be developed to provoke transitions that are not allowed by the software requirements.

6.4.3 Requirements-Based Testing Methods

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

a. Requirements-Based Hardware/Software Integration Testing: This testing method should concentrate on error sources associated with the software operating within the target computer environment, and on the high-level functionality. The objective of requirements-based hardware/software integration testing is to ensure that the software in the target computer will satisfy the high-level requirements. Typical errors revealed by this testing method include:

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

6.4.2.2 Jeux de test de robustesse

Les jeux de test de robustesse ont pour objectif de démontrer la capacité du logiciel à répondre à des données d'entrée et des conditions anormales. Les jeux de test de robustesse contiennent notamment :

a. On doit mettre en oeuvre des variables réelles et entières en prenant des valeurs des classes d'équivalence invalides.

b. L'initialisation du système doit se faire avec des conditions anormales.

c. On doit déterminer les modes de panne possibles des données d'entrée, en particulier les chaînes de données numériques complexes provenant d'un système extérieur.

d. Pour les boucles dont l'indice est une valeur calculée, on devra développer des jeux de test de manière à calculer des valeurs de l'indice de boucle extérieures au domaine de validité du compteur, et démontrer ainsi la robustesse du code lié aux boucles.

e. On doit vérifier que les mécanismes de protection contre les temps de calculs excessifs répondent correctement.

f. Pour les fonctions liées au temps, tels que filtres, intégrateurs, et retards, on doit développer des jeux de test pour vérifier les mécanismes de protection contre les débordements de capacité arithmétique.

g. Pour les transitions d'états, on doit développer des jeux de test afin de provoquer des transitions qui ne soient pas permises par les spécifications du logiciel.

6.4.3 Méthodes de test fondées sur les exigences

Les méthodes de test fondées sur les exigences comprennent les méthodes (fondées sur les exigences) de test d'intégration matériel/logiciel, de test d'intégration de logiciel, et de test de bas niveau. Sauf pour les tests d'intégration matériel/logiciel, ces méthodes n'imposent pas un environnement ou une stratégie de tests spécifiques. Les directives sont :

a. Tests d'intégration matériel/logiciel fondés sur les exigences : Cette méthode de test doit être concentrée sur les sources d'erreur associées au logiciel fonctionnant dans l'environnement de la machine cible, et sur la fonctionnalité de haut niveau. Le test d'intégration matériel/logiciel fondé sur les exigences a pour objectif de s'assurer que le logiciel, dans la machine cible, satisfait les exigences de haut niveau. Les erreurs typiques révélées par cette méthode de test sont les suivantes :

   - Mécanisme incorrect de traitement des interruptions.
   - Non respect des exigences de temps d'exécution.
   - Réponse incorrecte du logiciel à des transitoires ou des pannes du matériel, par exemple séquencement de mise en route, transitoires sur les flux d'entrées et transitoires d'alimentation.
   - Problèmes de bus de données et autres conflits d'accès aux ressources, par exemple d'organisation de la mémoire.
   - Inaptitude des tests intégrés à détecter des pannes.
   - Erreurs d'interfaces matériel/logiciel.
   - Comportement incorrect des boucles de retour d'information.
   - Contrôle incorrect du matériel de gestion mémoire ou d'autres dispositifs matériels sous contrôle logiciel.
   - Débordement de piles.
   - Fonctionnement incorrect du (ou des) mécanisme(s) employé(s) pour confirmer l'exactitude et la compatibilité du logiciel téléchargeable.
   - Violations du partitionnement du logiciel.

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

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

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

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

### 6.4.4 Test Coverage Analysis

Test coverage analysis is a two step process, involving requirements-based coverage analysis and structural coverage analysis. The first step analyzes the test cases in relation to the software requirements to confirm that the selected test cases satisfy the specified criteria. The second step confirms that the requirements-based test procedures exercised the code structure. Structural coverage analysis may not satisfy the specified criteria. Additional guidelines are provided for resolution of such situations as dead code (subparagraph 6.4.4.3).

#### 6.4.4.1 Requirements-Based Test Coverage Analysis

The objective of this analysis is to determine how well the requirements-based testing verified the implementation of the software requirements. This analysis may reveal the need for additional requirements-based test cases. The requirements-based test coverage analysis should show that:

a. Test cases exist for each software requirement.

b. Tests d'intégration du logiciel fondé sur les exigences : Cette méthode de test doit être concentrée sur les relations mutuelles entre les spécifications du logiciel et la mise en application de ces spécifications par l'architecture du logiciel. Le test d'intégration du logiciel fondé sur les exigences a pour objectif de s'assurer que les composants logiciel interagissent correctement les uns avec les autres et satisfont les spécifications du logiciel et son architecture. On peut employer cette méthode en étendant le domaine des exigences par intégration successive des composants du code et extension correspondante du domaine des jeux de test. Les erreurs typiques révélées par cette méthode de test sont :

- Initialisation incorrecte des variables et des constantes.
- Erreurs dans le passage de paramètres.
- Altération de données, en particulier de données globales.
- Résolution numérique de bout en bout inadéquate.
- Séquencement incorrect d'événements et d'opérations.

c. Tests de bas niveau fondés sur les exigences : Cette méthode de test doit être concentrée sur la démonstration du fait que chaque composant logiciel est conforme à ses exigences de bas niveau. Le test de bas niveau fondé sur les exigences a pour but de s'assurer que les composants logiciel satisfont leurs exigences de bas niveau. Les erreurs typiques révélées par cette méthode de test sont les suivantes :

- Inaptitude d'un algorithme à satisfaire une spécification du logiciel.
- Opérations de boucle incorrectes.
- Décisions logiques incorrectes.
- Inaptitude à traiter correctement des combinaisons valides de données d'entrée.
- Réponses incorrectes à des données d'entrée manquantes ou altérées.
- Mécanisme incorrect de traitement des exceptions, telles que des erreurs arithmétiques et violations de limites de tableau.
- Séquence de calcul incorrecte.
- Précision, justesse ou performances inadéquates d'un algorithme.

### 6.4.4 Analyse de la couverture de test

L'analyse de couverture de test est un processus en deux étapes, impliquant une analyse de couverture fondée sur les exigences et une analyse de la couverture structurelle. La première étape analyse les jeux de test en relation avec les spécifications du logiciel de manière à confirmer que les jeux de test choisis satisfont les critères spécifiés. La seconde étape confirme que les procédures de test fondées sur les exigences ont mis en oeuvre la structure du code. L'analyse de couverture structurelle peut ne pas satisfaire les critères spécifiés. Des recommandations complémentaires sont fournies pour résoudre de telles situations comme le code mort (sous-paragraphe 6.4.4.3).

#### 6.4.4.1 Analyse de la couverture de test fondée sur les exigences

Cette analyse a pour objectif de déterminer dans quelle mesure le test fondé sur les exigences vérifie la mise en application des spécifications du logiciel. Cette analyse peut révéler la nécessité de jeux de test supplémentaires fondés sur les exigences. L'analyse de couverture de test fondé sur les exigences doit montrer que :

a. Des jeux de test existent pour chaque spécification du logiciel.

b. Test cases satisfy the criteria of normal and robustness testing as defined in paragraph 6.4.2.

6.4.4.2 Structural Coverage Analysis

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

a. The analysis should confirm the degree of structural coverage appropriate to the software level.

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

c. The analysis should confirm the data coupling and control coupling between the code components.

6.4.4.3 Structural Coverage Analysis Resolution

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

a. Shortcomings in requirements-based test cases or procedures: The test cases should be supplemented or test procedures changed to provide the missing coverage. The method(s) used to perform the requirements-based coverage analysis may need to be reviewed.

b. Inadequacies in software requirements: The software requirements should be modified and additional test cases developed and test procedures executed.

c. Dead code: The code should be removed and an analysis performed to assess the effect and the need for reverification.

d. Deactivated code: For deactivated code which is not intended to be executed in any configuration used within an aircraft or engine, a combination of analysis and testing should show that the means by which such code could be inadvertently executed are prevented, isolated, or eliminated. For deactivated code which is only executed in certain configurations of the target computer environment, the operational configuration needed for normal execution of this code should be established and additional test cases and test procedures developed to satisfy the required coverage objectives.

b. Les jeux de test satisfont les critères de tests dans les plages de variation prévues et de tests de robustesse comme défini dans le paragraphe 6.4.2.

6.4.4.2 Analyse de la couverture structurelle

Cette analyse a pour objectif de déterminer quelles structures de code n'ont pas été mises en oeuvre par les procédures de test fondé sur les exigences. Les jeux de test fondés sur les exigences peuvent ne pas avoir complètement mis en oeuvre la structure du code, de sorte qu'on fait une analyse de la couverture structurelle et des vérifications complémentaires pour obtenir cette couverture. Les directives sont les suivantes :

a. L'analyse doit confirmer le degré de couverture structurelle approprié au niveau logiciel.

b. On peut réaliser l'analyse de couverture structurelle sur le Code Source, sauf si le logiciel est de Niveau A et si le compilateur génère un code objet qui n'est pas directement traçable vers les instructions du Code Source. On doit alors procéder à une vérification complémentaire sur le code objet afin d'établir l'exactitude de ces séquences de code générées. Un contrôle des limites de tableau généré par le compilateur dans le code objet, constitue un exemple de code objet qui n'est pas directement traçable vers le Code Source.

c. L'analyse doit confirmer le couplage par le flux de données et le couplage par le flux de contrôle entre les composants du code.

6.4.4.3 Résolution de l'analyse de la couverture structurelle

L'analyse de couverture structurelle peut révéler qu'une structure du code n'a pas été mise en oeuvre au cours des tests. La résolution peut exiger une activité complémentaire du processus de vérification du logiciel. Cette structure du code non exécutée peut être le résultat de :

a. Imperfections dans les jeux ou procédures de test fondés sur les exigences : On doit enrichir les jeux de test ou modifier les procédures de test afin d'obtenir la couverture manquante. Il pourra être nécessaire de revoir la (ou les) méthode(s) utilisée(s) pour réaliser l'analyse de couverture fondée sur les exigences.

b. Insuffisances dans les spécifications du logiciel : On doit modifier ces spécifications, développer des jeux de test, et exécuter des procédures de test.

c. Code mort : On doit éliminer ce code et procéder à une analyse pour en évaluer l'effet et le besoin d'une revérification.

d. Code désactivé : Pour un code désactivé dont l'exécution n'est prévue dans aucune configuration employée dans un aéronef ou un moteur, une combinaison d'analyse et de test doit montrer qu'on a évité, isolé, ou éliminé les moyens pouvant provoquer par inadvertance une exécution de ce code. Pour un code désactivé qui n'est exécuté que dans certaines configurations de l'environnement de la machine cible, on doit établir la configuration opérationnelle nécessaire à l'exécution normale de ce code et développer des jeux et procédures de test destinés à satisfaire les objectifs de couverture requis.

# SECTION 7

# SOFTWARE CONFIGURATION MANAGEMENT PROCESS

This section discusses the objectives and activities of the software configuration management (SCM) process. The SCM process is applied as defined by the software planning process (section 4) and the Software Configuration Management Plan (subsection 11.4). Outputs of the SCM process are recorded in Software Configuration Management Records (subsection 11.18) or in other software life cycle data.

Table A-8 of Annex A is a summary of the objectives and outputs of the SCM process.

## 7.1 SOFTWARE CONFIGURATION MANAGEMENT PROCESS OBJECTIVES

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

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

b. Provide the ability to consistently replicate the Executable Object Code for software manufacture or to regenerate it in case of a need for investigation or modification.

c. Provide control of process inputs and outputs during the software life cycle that ensures consistency and repeatability of process activities.

d. Provide a known point for review, assessing status, and change control by control of configuration items and the establishment of baselines.

e. Provide controls that ensure problems receive attention and changes are recorded, approved, and implemented.

f. Provide evidence of approval of the software by control of the outputs of the software life cycle processes.

g. Aid the assessment of the software product compliance with requirements.

h. Ensure that secure physical archiving, recovery and control are maintained for the configuration items.

The objectives for SCM are independent of software level. However, two categories of software life cycle data may exist based on the SCM controls applied to the data (subsection 7.3).

## 7.2 SOFTWARE CONFIGURATION MANAGEMENT PROCESS ACTIVITIES

The SCM process includes the activities of configuration identification, change control, baseline establishment, and archiving of the software product, including the related software life cycle data. The following guidelines define the objectives for each SCM process activity. The SCM process does not stop when the software product is accepted by the certification authority, but continues throughout the service life of the airborne system or equipment.

# SECTION 7

# GESTION DE CONFIGURATION DU LOGICIEL

Cette section traite des objectifs et des activités du processus de gestion de configuration du logiciel (SCM). Le processus de SCM s'applique de la manière définie par la planification du logiciel (section 4) et par le Plan de Gestion de Configuration du Logiciel (sous-section 11.4). Les produits du processus de SCM sont enregistrés dans les Documents de Gestion de Configuration du Logiciel (sous-section 11.18) ou dans d'autres données du cycle de vie du logiciel.

Le Tableau A-8 de l'Annexe A constitue un résumé des objectifs et des produits du processus de SCM.

## 7.1 OBJECTIFS DU PROCESSUS DE GESTION DE CONFIGURATION DU LOGICIEL

Le processus de SCM, agissant en coopération avec les autres processus du cycle de vie du logiciel, aide à satisfaire les objectifs généraux en :

a. Fournissant une configuration définie et contrôlée du logiciel tout au long de son cycle de vie.

b. Fournissant la capacité de reproduire de manière cohérente le Code Objet Exécutable à des fins de production du logiciel ou de le régénérer pour des besoins d'investigation ou de modification.

c. Assurant la maîtrise des entrées et des produits des processus au cours du cycle de vie du logiciel qui garantit la cohérence et la reproductibilité des activités des processus.

d. Fournissant une référence connue pour l'examen, l'évaluation de l'état et la gestion des modifications grâce à la maîtrise des éléments de configuration et la création de référentiels.

e. Fournissant des moyens de contrôle qui garantissent que les problèmes sont correctement étudiés et que les modifications sont enregistrées, approuvées et réalisées.

f. Apportant des preuves d'approbation du logiciel à travers la maîtrise des produits de son cycle de vie.

g. Aidant à l'évaluation de la conformité du produit logiciel avec les exigences.

h. Garantissant l'entretien d'un archivage physique, et l'existence d'une restauration et d'un contrôle sûrs pour les éléments de configuration.

Les objectifs de SCM sont indépendants du niveau logiciel. Cependant, il peut exister deux catégories de données de cycle de vie du logiciel, selon les contrôles SCM appliqués aux données (sous-section 7.3).

## 7.2 ACTIVITES DU PROCESSUS DE GESTION DE CONFIGURATION DU LOGICIEL

Le processus de SCM comprend les activités d'identification de configuration, de gestion des modifications, d'établissement des référentiels, et d'archivage du produit logiciel, y compris les données associées à son cycle de vie. Les recommandations ci-dessous définissent les objectifs de chaque activité du processus de SCM. Le processus de SCM ne s'arrête pas quand le produit logiciel est accepté par l'Autorité de certification, mais se poursuit tout au long de la vie en service du système ou équipement de bord.

7.2.1 Configuration Identification

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

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

b. Configuration identification should be established for each configuration item, for each separately controlled component of a configuration item, and for combinations of configuration items that comprise a software product.

c. Configuration items should be configuration-identified prior to the implementation of change control and traceability data recording.

d. A configuration item should be configuration-identified before that item is used by other software life cycle processes, referenced by other software life cycle data, or used for software manufacture or software loading.

e. If the software product identification cannot be determined by physical examination (for example, part number plate examination), then the Executable Object Code should contain configuration identification which can be accessed by other parts of the airborne system or equipment. This may be applicable for field-loadable software (subsection 2.5).

7.2.2 Baselines and Traceability

The objective of baseline establishment is to define a basis for further software life cycle process activity and allow reference to, control of, and traceability between configuration items. Guidance includes:

a. Baselines should be established for configuration items used for certification credit. (Intermediate baselines may be established to aid in controlling software life cycle process activities.)

b. A software product baseline should be established for the software product and defined in the Software Configuration Index (subsection 11.16).

*NOTE: User-modifiable software is not included in the software product baseline, except for its associated protection and boundary components. Therefore, modifications may be made to user-modifiable software without affecting the configuration identification of the software product baseline.*

c. Baselines should be established in controlled software libraries (physical, electronic, or other) to ensure their integrity. Once a baseline is established, it should be protected from change.

d. Change control activities should be followed to develop a derivative baseline from an established baseline.

e. A baseline should be traceable to the baseline from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous baseline.

f. A configuration item should be traceable to the configuration item from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous configuration item.

g. A baseline or configuration item should be traceable either to the output it identifies or to the process with which it is associated.

7.2.1 Identification de configuration

L'activité d'identification de configuration a pour objectif de repérer sans ambiguïté chaque élément de configuration (et ses versions successives) de manière à constituer une base pour la gestion et la référence des éléments de configuration. Les directives sont :

a. L'identification de configuration doit être établie pour les données du cycle de vie du logiciel.

b. L'identification de configuration doit être établie pour chaque élément de configuration, pour chaque composant d'un élément de configuration géré séparément, et pour des combinaisons d'éléments de configuration qui constituent un produit logiciel.

c. Les éléments de configuration doivent être identifiés en configuration avant la mise en oeuvre de la gestion des modifications et de l'enregistrement des données de traçabilité.

d. Un élément de configuration doit être identifié en configuration avant d'être utilisé par d'autres processus du cycle de vie, pris comme référence par d'autres données du cycle de vie, ou utilisé pour la production ou le chargement du logiciel.

e. S'il n'est pas possible d'identifier un produit logiciel par examen physique (par exemple, examen de la plaque d'identification), le Code Objet Exécutable doit alors comporter une identification de configuration à laquelle pourront accéder d'autres éléments du système ou équipement de bord. Ceci peut s'appliquer aux logiciels téléchargeables (sous-section 2.5).

7.2.2 Référentiels et traçabilité

L'activité d'établissement des référentiels a pour objectif de définir une base pour les activités ultérieures du cycle de vie du logiciel, de permettre de se référer aux éléments de configuration, de les gérer et de permettre la traçabilité entre éléments. Les directives sont :

a. Des référentiels doivent être établis pour les éléments de configuration utilisés pour obtenir la certification. (L'établissement de référentiels intermédiaires peut contribuer à la maîtrise des activités du cycle de vie du logiciel).

b. Un référentiel final du produit logiciel doit être établi. Il doit être défini dans le Répertoire de la Configuration du Logiciel (sous-section 11.16).

*NOTA : Ce référentiel ne comporte pas de logiciel modifiable par l'utilisateur, mais comporte les composants de protection et les composants d'interface correspondants. Par conséquent, on peut procéder à des modifications du logiciel modifiable par l'utilisateur sans affecter l'identification de configuration du référentiel final.*

c. Les référentiels doivent être établis dans des bibliothèques (physiques, électroniques, ou autres) sous contrôle pour assurer leur intégrité. Une fois un référentiel établi, il doit être protégé contre les modifications.

d. Des procédures de gestion des modifications doivent être appliquées pour développer un référentiel dérivé à partir d'un référentiel établi.

e. Un référentiel doit être traçable vers le référentiel dont il est dérivé, lorsqu'un crédit de certification est recherché pour des activités ou des données du cycle de vie associées au développement du référentiel antérieur.

f. Un élément de configuration doit être traçable vers l'élément de configuration dont il est dérivé, lorsqu'un crédit de certification est recherché pour des activités ou des données du cycle de vie associées au développement de l'élément de configuration antérieur.

g. Un référentiel ou un élément de configuration doit être traçable soit vers le produit qu'ils identifient, soit vers le processus auquel ils sont associés.

### 7.2.3 Problem Reporting, Tracking and Corrective Action

The objective of problem reporting, tracking and corrective action is to record process non-compliance with software plans and standards, to record deficiencies of outputs of software life cycle processes, to record anomalous behavior of software products, and to ensure resolution of these problems.

*NOTE: Software life cycle process and software product problems may be recorded in separate problem reporting systems.*

Guidance includes:

a. A problem report should be prepared that describes the process non-compliance with plans, output deficiency, or software anomalous behavior, and the corrective action taken, as defined in subsection 11.17.

b. Problem reporting should provide for configuration identification of affected configuration item(s) or definition of affected process activities, status reporting of problem reports, and approval and closure of problem reports.

c. Problem reports that require corrective action of the software product or outputs of software life cycle processes should invoke the change control activity.

*NOTE: The problem reporting and change control activities are related.*

### 7.2.4 Change Control

The objective of the change control activity is to provide for recording, evaluation, resolution and approval of changes throughout the software life cycle. Guidance includes:

a. Change control should preserve the integrity of the configuration items and baselines by providing protection against their change.

b. Change control should ensure that any change to a configuration item requires a change to its configuration identification.

c. Changes to baselines and to configuration items under change control should be recorded, approved, and tracked. Problem reporting is related to change control, since resolution of a reported problem may result in changes to configuration items or baselines.

   *NOTE: It is generally recognized that early implementation of change control assists the control and management of software life cycle process activities.*

d. Software changes should be traced to their origin and the software life cycle processes repeated from the point at which the change affects their outputs. For example, an error discovered at hardware/software integration, that is shown to result from an incorrect design, should result in design correction, code correction and repetition of the associated integral process activities.

e. Throughout the change activity, software life cycle data affected by the change should be updated and records should be maintained for the change control activity.

The change control activity is aided by the change review activity.

7.2.3 Compte rendu des anomalies, suivi et actions correctives

Les activités de compte rendu, de suivi des anomalies et des actions correctives ont pour objectif d'enregistrer les non-conformités des processus vis à vis des plans et règles du logiciel, les insuffisances des produits du cycle de vie du logiciel, les dysfonctionnements du logiciel, et de garantir la résolution de ces problèmes.

*NOTA : Il est possible d'enregistrer dans des systèmes de compte rendu séparés les anomalies concernant les processus du cycle de vie du logiciel et celles concernant le produit logiciel.*

Les directives sont :

a. Un rapport d'anomalie doit être établi pour décrire la non-conformité du processus avec les plans, l'insuffisance du produit d'un processus ou le dysfonctionnement du logiciel, ainsi que l'action corrective exécutée, de la manière définie dans la sous-section 11.17.

b. Le compte rendu des anomalies doit permettre d'identifier le (ou les) élément(s) de configuration ou les activités des processus mis en cause, il doit aussi permettre de suivre l'état des rapports d'anomalie, de les approuver et de les clore.

c. Les rapports d'anomalies nécessitant une action corrective sur le produit logiciel ou sur d'autres produits du cycle de vie doivent faire appel à la gestion des modifications.

*NOTA : Les activités de compte rendu des anomalies et de gestion des modifications sont liées.*

7.2.4 Gestion des modifications

L'activité de gestion des modifications a pour objectif d'enregistrer, d'évaluer, de réaliser et d'approuver des modifications tout au long du cycle de vie du logiciel. Les directives sont :

a. La gestion des modifications doit préserver l'intégrité des éléments de configuration et des référentiels en assurant leur protection contre les modifications.

b. La gestion des modifications doit garantir que toute modification d'un élément de configuration implique nécessairement une modification de son identification de configuration.

c. Les modifications apportées aux référentiels et aux éléments de configuration soumis à la gestion des modifications doivent être enregistrées, approuvées, et faire l'objet d'un suivi. Le compte rendu des anomalies est lié à la gestion des modifications, car la résolution d'une anomalie peut entraîner des modifications des éléments de configuration ou des référentiels.

*NOTA : Il est en général admis que la mise en oeuvre précoce de la gestion des modifications contribue à la maîtrise et à la gestion des activités du cycle de vie du logiciel.*

d. On doit pouvoir remonter à la cause initiale des modifications, et les activités des processus du cycle de vie, dont les résultats sont affectés par la modification, doivent être reprises. Par exemple, une erreur découverte lors de l'intégration matériel/logiciel, et dont il est montré qu'elle résulte d'une conception incorrecte, doit entraîner une correction de la conception et du code et la reprise des activités des processus intégraux associés.

e. Pendant toute l'activité de modification, on doit actualiser les données du cycle de vie du logiciel affectées par la modification, et conserver des enregistrements de l'activité de gestion des modifications.

L'activité de revue des modifications assiste l'activité de gestion des modifications.

7.2.5 Change Review

The objective of the change review activity is to ensure problems and changes are assessed, approved or disapproved, approved changes are implemented, and feedback is provided to affected processes through problem reporting and change control methods defined during the software planning process. The change review activity should include:

a. Confirmation that affected configuration items are configuration identified.

b. Assessment of the impact on safety-related requirements with feedback to the system safety assessment process

c. Assessment of the problem or change, with decisions for action to be taken.

d. Feedback of problem report or change impact and decisions to affected processes.

7.2.6 Configuration Status Accounting

The objective of the status accounting activity is to provide data for the configuration management of software life cycle processes with respect to configuration identification, baselines, problem reports, and change control. The status accounting activity should include:

a. Reporting on configuration item identification, baseline identification, problem report status, change history, and release status.

b. Definition of the data to be maintained and the means of recording and reporting status of this data.

7.2.7 Archive, Retrieval and Release

The objective of the archive and retrieval activity is to ensure that the software life cycle data associated with the software product can be retrieved in case of a need to duplicate, regenerate, retest or modify the software product. The objective of the release activity is to ensure that only authorized software is used, especially for software manufacturing, in addition to being archived and retrievable. Guidance includes:

a. Software life cycle data associated with the software product should be retrievable from the approved source (for example, the developing organization or company).

b. Procedures should be established to ensure the integrity of the stored data (regardless of medium of storage) by:

   (1) Ensuring that no unauthorized changes can be made.

   (2) Selecting storage media that minimize regeneration errors or deterioration.

   (3) Exercising and/or refreshing archived data at a frequency compatible with the storage life of the medium.

   (4) Storing duplicate copies in physically separate archives that minimize the risk of loss in the event of a disaster.

c. The duplication process should be verified to produce accurate copies, and procedures should exist that ensure error-free copying of the Executable Object Code.

7.2.5 Revue des modifications

L'activité de revue des modifications a pour objectif de garantir que les anomalies et les modifications sont évaluées et approuvées ou rejetées, que les modifications approuvées sont réalisées, et que les méthodes de compte rendu des anomalies et de gestion des modifications définies durant la planification du logiciel assurent un retour d'information vers les processus mis en cause. L'activité de revue des modifications doit comporter :

a. La confirmation que les éléments de configuration mis en cause sont identifiés.

b. L'évaluation de l'impact sur les exigences liées à la sécurité avec un retour vers l'analyse de sécurité du système.

c. L'analyse de l'anomalie ou de la modification, avec les décisions sur les actions à entreprendre.

d. Le retour d'information relatif à l'impact de l'anomalie ou de la modification, ainsi qu'aux décisions associées, vers les processus mis en cause.

7.2.6 Suivi des états de configuration

L'activité de suivi des états de configuration a pour objectif de fournir des données de gestion de configuration relatives à l'identification de la configuration, aux référentiels, aux rapports d'anomalies et à la gestion des modifications. Cette activité doit comporter :

a. Le compte rendu relatif à l'identification des éléments de configuration, l'identification des référentiels, l'état des rapports d'anomalies, l'historique des modifications et l'état du produit mis à disposition.

b. La définition des données à tenir à jour et des moyens pour enregistrer et rendre compte de leur état.

7.2.7 Archivage, restauration et mise à disposition

L'activité d'archivage et de restauration a pour objectif de garantir que les données du cycle de vie associées au produit logiciel peuvent être restaurées pour reproduire, régénérer, tester de nouveau, ou modifier ce produit. L'activité de mise à disposition a pour objectif de garantir de plus, que seul le logiciel autorisé est utilisé, en particulier à des fins de production. Les directives sont :

a. Les données du cycle de vie associées au produit logiciel doivent pouvoir être récupérées auprès de la source approuvée (par exemple, l'Organisation ou la Société qui les a développées).

b. Des procédures qui garantissent l'intégrité des données conservées (indépendamment du moyen de stockage) doivent être élaborées en :

   (1) Garantissant qu'aucune modification non autorisée ne peut être effectuée.

   (2) Choisissant des moyens de stockage qui minimisent les erreurs de régénération ou les détériorations.

   (3) Vérifiant et/ou rafraîchissant les données archivées à une fréquence compatible avec la durée de stockage du support utilisé.

   (4) Stockant les copies dans des lieux d'archivage physiquement séparés, afin de réduire le risque de perte en cas de sinistre.

c. Le processus de duplication doit être vérifié de manière à produire des copies exactes, et des procédures doivent exister garantissant que la copie du Code Objet Exécutable s'effectue sans erreur.

d. Configuration items should be identified and released prior to use for software manufacture and the authority for their release should be established. As a minimum, the components of the software product loaded into the airborne system or equipment (which includes the Executable Object Code and may also include associated media for software loading) should be released.

*NOTE: Release is generally also required for the data that defines the approved software for loading into the airborne system or equipment. Definition of that data is outside the scope of this document, but may include the Software Configuration Index.*

e. Data retention procedures should be established to satisfy airworthiness requirements and enable software modifications.

*NOTE: Additional data retention considerations may include items such as business needs and future certification authority reviews, which are outside the scope of this document.*

### 7.2.8 Software Load Control

The objective of the software load control activity is to ensure that the Executable Object Code is loaded into the airborne system or equipment with appropriate safeguards. Software load control refers to the process by which programmed instructions and data are transferred from a master memory device into the airborne system or equipment. For example, methods may include (subject to approval by the certification authority) the installation of factory pre-programmed memory devices or in situ re-programming of the airborne system or equipment using a field loading device. Whichever method is used, software load control should include:

a. Procedures for part numbering and media identification that identify software configurations that are intended to be approved for loading into the airborne system or equipment.

b. Whether the software is delivered as an end item or is delivered installed in the airborne system or equipment, records should be kept that confirm software compatibility with the airborne system or equipment hardware.

*NOTE: Additional guidance on software load control is provided in subsection 2.5.*

### 7.2.9 Software Life Cycle Environment Control

The objective of software life cycle environment control is to ensure that the tools used to produce the software are identified, controlled, and retrievable. The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (subsection 11.15). Guidance includes:

a. Configuration identification should be established for the Executable Object Code (or equivalent) of the tools used to develop, control, build, verify, and load the software.

b. The SCM process for controlling qualified tools, should comply with the objectives associated with Control Category 1 or 2 data (subsection 7.3), as specified in paragraph 12.2.3, item b.

c. Unless 7.2.9 item b applies, the SCM process for controlling the Executable Object Code (or equivalent) of tools used to build and load the software (for example, compilers, assemblers, linkage editors) should comply with the objectives associated with Control Category 2 data, as a minimum.

d. Les éléments de configuration doivent être identifiés et mis à disposition avant d'être utilisés pour produire le logiciel et l'autorité responsable de la mise à disposition doit être définie. Au minimum, les composants du produit logiciel chargé dans le système ou équipement de bord (incluant le Code Objet Exécutable mais pouvant comprendre aussi les moyens utilisés pour charger le logiciel) doivent être mis à disposition.

*NOTA : En général, les données définissant le logiciel approuvé pour le chargement du logiciel d'un système ou équipement de bord doivent être également mises à disposition. La définition de ces données est hors du cadre de ce document, mais peut comprendre un Répertoire de la Configuration du Logiciel.*

e. Des procédures de conservation des données doivent être élaborées pour satisfaire les exigences de navigabilité et permettre les modifications du logiciel.

*NOTA : La conservation des données peut obéir à des considérations complémentaires comme des besoins économiques et des revues futures par l'Autorité de certification, qui sont hors du cadre de ce document.*

### 7.2.8 Contrôle du chargement du logiciel

L'activité de contrôle du chargement du logiciel a pour objectif de garantir que le Code Objet Exécutable est chargé dans le système ou l'équipement de bord en utilisant des protections appropriées. Le contrôle du chargement du logiciel concerne le processus par lequel des instructions et des données programmées sont transférées d'un dispositif de mémoire étalon vers le système ou équipement de bord. Par exemple les méthodes peuvent comprendre (sous réserve de l'approbation par l'Autorité de certification) l'installation d'éléments mémoire pré-programmés en usine ou la reprogrammation in situ du système ou équipement de bord au moyen d'un dispositif de téléchargement. Quelle que soit la méthode utilisée, le contrôle du chargement du logiciel doit comprendre :

a. Des procédures de numérotation des éléments et d'identification des supports qui référencent les configurations de logiciel destinées à être approuvées en vue de leur chargement dans le système ou équipement de bord.

b. Des documents confirmant la compatibilité du logiciel avec le matériel du système ou équipement de bord qui doivent être conservés, que le logiciel soit livré comme produit final ou chargé dans le système ou équipement de bord.

*NOTA : La sous-section 2.5 fournit des directives complémentaires pour le contrôle du chargement du logiciel.*

### 7.2.9 Contrôle de l'environnement de développement du logiciel

Le contrôle de l'environnement de développement du logiciel a pour objectif de garantir que les outils utilisés pour générer le logiciel sont identifiés, gérés, et qu'il est possible de les restaurer. Les outils de l'environnement de développement du logiciel sont définis par le processus de planification du logiciel et identifiés dans le Répertoire de la Configuration de l'Environnement du Logiciel (sous-section 11.15). Les directives sont :

a. L'identification de configuration doit être établie pour le Code Objet Exécutable (ou équivalent) des outils utilisés pour développer, gérer, générer, vérifier, et charger le logiciel.

b. Le processus de SCM mis en oeuvre pour gérer les outils qualifiés doit être conforme aux objectifs associés aux données de Catégorie de Contrôle 1 ou 2 (sous-section 7.3), de la manière définie dans le paragraphe 12.2.3 b.

c. Si le paragraphe 7.2.9 b ne s'applique pas, le processus de SCM mis en oeuvre pour gérer le Code Objet Exécutable (ou équivalent) des outils utilisés pour générer et charger le logiciel (par exemple, les compilateurs, assembleurs, éditeurs de liens) doit être au minimum conforme aux objectifs associés aux données de Catégorie de Contrôle 2.

## 7.3 DATA CONTROL CATEGORIES

Software life cycle data can be assigned to one of two categories: Control Category 1 (CC1) and Control Category 2 (CC2). These categories are related to the configuration management controls placed on the data. Table 7-1 defines the set of SCM process objectives associated with each control category, where ø indicates that the objectives apply for software life cycle data of that category.

The tables of Annex A specify the control category for each software life cycle data item, by software level. Guidance for data control categories includes:

a. The SCM process objectives for software life cycle data categorized as CC1 should be applied according to Table 7-1.

b. The SCM process objectives for software life cycle data categorized as CC2 should be applied according to Table 7-1 as a minimum.

| SCMProcess Objective | Reference | CC1 | CC2 |
|---|---|---|---|
| Configuration Identification | 7.2.1 | ø | ø |
| Baselines | 7.2.2 a, b, c, d, e | ø | |
| Traceability | 7.2.2f, g | ø | ø |
| Problem Reporting | 7.2.3 | ø | |
| Change Control - integrity and identification | 7.2.4a, b | ø | ø |
| Change Control - tracking | 7.2.4c, d, e | ø | |
| Change Review | 7.2.5 | ø | |
| Configuration Status Accounting | 7.2.6 | ø | |
| Retrieval | 7.2.7a | ø | ø |
| Protection against Unauthorized Changes | 7.2.7b(1) | ø | ø |
| Media Selection, Refreshing, Duplication | 7.2.7b(2), (3), (4), c | ø | |
| Release | 7.2.7d | ø | |
| Data Retention | 7. 2.7e | ø | ø |

TABLE 7-1: SCM PROCESS OBJECTIVES ASSOCIATED WITH CC1 AND CC2 DATA

## 7.3 CATEGORIES DE CONTROLE DES DONNEES

Les données du cycle de vie du logiciel peuvent être classées en deux catégories : Catégorie de Contrôle 1 (CC1) et Catégorie de Contrôle 2 (CC2). Ces catégories sont liées aux règles de gestion de configuration appliquées aux données. Le Tableau 7-1 définit les objectifs du processus de SCM associés à chaque catégorie de contrôle, le signe ° indiquant si les objectifs s'appliquent aux données du cycle de vie de cette catégorie.

Les tableaux de l'Annexe A déterminent la catégorie de contrôle pour chaque unité d'information, par niveau logiciel. Les directives sont :

a. Les objectifs du processus de SCM pour les données du cycle de vie de la catégorie CC1 doivent être conformes au Tableau 7-1.

b. Les objectifs du processus de SCM pour les données du cycle de vie de la catégorie CC2 doivent au minimum être conformes au Tableau 7-1.

| **Objectif du processus de SCM** | **Reference** | **CC1** | **CC2** |
|---|---|---|---|
| Identification de configuration | 7.2.1 | ○ | ○ |
| Référentiels | 7.2.2 a, b, c, d, e | ○ | |
| Traçabilité | 7.2.2f, g | ○ | ○ |
| Compte rendu des anomalies | 7.2.3 | ○ | |
| Intégrité et identification de la gestion des modifications | 7.2.4a, b | ○ | ○ |
| Suivi de la gestion des modifications | 7.2.4c, d, e | ○ | |
| Revue des modifications | 7.2.5 | ○ | |
| Suivi des états de configuration | 7.2.6 | ○ | |
| Restauration | 7.2.7a | ○ | ○ |
| Protection contre les modification non autorisées | 7.2.7b(1) | ○ | ○ |
| Choix des supports rafraîchissement, copie | 7.2.7b(2), (3), (4), c | ○ | |
| Mise à disposition | 7.2.7d | ○ | |
| Conservation des données | 7. 2.7e | ○ | ○ |

TABLEAU 7-1 : OBJECTIFS DU PROCESSUS DE SCM ASSOCIES AUX DONNEES CC1 ET CC2

# SECTION 8

# SOFTWARE QUALITY ASSURANCE PROCESS

This section discusses the objectives and activities of the software quality assurance (SQA) process. The SQA process is applied as defined by the software planning process (section 4) and the Software Quality Assurance Plan (subsection 11.5). Outputs of the SQA process activities are recorded in Software Quality Assurance Records (subsection 11.19) or other software life cycle data.

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

## 8.1 SOFTWARE QUALITY ASSURANCE PROCESS OBJECTIVES

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

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

a. Software development processes and integral processes comply with approved software plans and standards.

b. The transition criteria for the software life cycle processes are satisfied.

c. A conformity review of the software product is conducted.

The applicability of the objectives by software level is specified in Table A-9 of Annex A.

## 8.2 SOFTWARE QUALITY ASSURANCE PROCESS ACTIVITIES

To satisfy the SQA process objectives:

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

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

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

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

   (1) Software plans are available as specified in subsection 4.2.

   (2) Deviations from the software plans and standards are detected, recorded, evaluated, tracked and resolved.

   *NOTE: It is generally accepted that early detection of process deviations assists efficient achievement of software life cycle process objectives.*

   (3) Approved deviations are recorded.

   (4) The software development environment has been provided as specified in the software plans.

# SECTION 8

# ASSURANCE QUALITE DU LOGICIEL

Cette section traite des objectifs et des activités du processus d'assurance qualité du logiciel (SQA). Le processus de SQA s'applique de la manière définie par la planification (section 4) et par le Plan d'Assurance Qualité du Logiciel (sous-section 11.5). Les produits des activités de SQA sont enregistrés dans les Documents d'Assurance Qualité du Logiciel (sous-section 11.19) ou dans d'autres données du cycle de vie du logiciel.

Le processus de SQA évalue le cycle de vie du logiciel et ses produits afin d'obtenir l'assurance que les objectifs sont satisfaits, que les insuffisances sont détectées, évaluées, suivies, et résolues, et que le produit logiciel et les données de son cycle de vie sont conformes aux exigences de certification.

## 8.1 OBJECTIFS DU PROCESSUS D'ASSURANCE QUALITE DU LOGICIEL

Les objectifs du processus de SQA sont d'assurer que le cycle de vie produit un logiciel conforme à ses spécifications en garantissant que les processus du cycle de vie sont réalisés en accord avec les plans et règles approuvés.

Le processus de SQA a pour objectif d'obtenir l'assurance que :

a. Le développement du logiciel et les processus intégraux sont conformes aux plans et règles approuvés.

b. Les critères de transition pour chaque processus du cycle de vie du logiciel sont satisfaits.

c. Une revue de conformité du produit logiciel est menée.

Le Tableau A-9 de l'Annexe A spécifie l'applicabilité des objectifs par niveau logiciel.

## 8.2 ACTIVITES DU PROCESSUS D'ASSURANCE QUALITE DU LOGICIEL

Afin de satisfaire les objectifs du processus de SQA :

a. Le processus de SQA doit jouer un rôle actif dans les activités du cycle de vie du logiciel. Ceux qui sont en charge de l'assurance qualité devront être investis de l'autorité, la responsabilité et l'indépendance permettant de garantir la satisfaction de ces objectifs.

b. Le processus de SQA doit fournir l'assurance que les plans et règles du logiciel ont été développés et soumis à revue pour en assurer la cohérence.

c. Le processus de SQA doit donner l'assurance que le cycle de vie du logiciel est conforme aux plans et règles approuvés.

d. Le processus de SQA doit comprendre des audits du processus de développement du logiciel et des processus intégraux au cours du cycle de vie du logiciel afin d'obtenir l'assurance que :

   (1) Les plans du logiciel sont disponibles comme indiqué dans la sous-section 4.2.

   (2) Les écarts avec les plans et règles du logiciel sont détectés, enregistrés, évalués, suivis et résolus.

   NOTA : *Il est généralement admis que la détection précoce des écarts de processus contribue efficacement à atteindre les* objectifs du cycle de vie du logiciel.

   (3) Les écarts approuvés sont enregistrés.

   (4) L'environnement de développement du logiciel a bien été fourni comme le spécifient les plans.

(5) The problem reporting, tracking and corrective action process complies with the Software Configuration Management Plan.

(6) Inputs provided to the software life cycle processes by the on-going system safety assessment process have been addressed.

NOTE: *Monitoring of the activities of software life cycle processes may be performed to provide assurance that the activities are under control.*

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

f. The SQA process should provide assurance that software life cycle data is controlled in accordance with the control categories as defined in subsection 7.3 and the tables of Annex A..

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

h. The SQA process should produce records of the SQA process activities (subsection 11.19), including audit results and evidence of completion of the software conformity review for each software product submitted as part of certification application.

## 8.3 SOFTWARE CONFORMITY REVIEW

The objective of the software conformity review is to obtain assurances, for a software product submitted as part of a certification application, that the software life cycle processes are complete, software life cycle data is complete, and the Executable Object Code is controlled and can be regenerated.

This review should determine that:

a. Planned software life cycle process activities for certification credit, including the generation of software life cycle data, have been completed and records of their completion are retained.

b. Software life cycle data developed from specific system requirements, safety-related requirements, or software requirements are traceable to those requirements.

c. Software life cycle data complies with software plans and standards, and is controlled in accordance with the SCM Plan.

d. Problem reports comply with the SCM Plan, have been evaluated and have their status recorded.

e. Software requirement deviations are recorded and approved.

f. The Executable Object Code can be regenerated from the archived Source Code.

g. The approved software can be loaded successfully through the use of released instructions.

h. Problem reports deferred from a previous software conformity review are re-evaluated to determine their status.

i. If certification credit is sought for the use of previously developed software, the current software product baseline is traceable to the previous baseline and the approved changes to that baseline.

NOTE: *For post-certification software modifications, a subset of the software conformity review activities, as justified by the significance of the change, may be performed.*

(5) Le processus de compte rendu et de suivi des anomalies, et celui des actions correctives sont conformes au Plan de Gestion de Configuration du Logiciel.

(6) Les entrées fournies au cycle de vie du logiciel par l'analyse de sécurité du système sont prises en compte.

*NOTA : Une surveillance continue des activités du cycle de vie du logiciel peut permettre d'obtenir l'assurance que ces activités sont maîtrisées.*

e. Le processus de SQA doit fournir l'assurance que les critères de transition ont été satisfaits conformément aux plans approuvés.

f. Le processus de SQA doit fournir l'assurance que les données du cycle de vie sont gérées en configuration en accord avec les catégories de contrôle définies dans la sous-section 7.3 et les tableaux de l'annexe A.

g. Avant la livraison d'un logiciel présenté dans le cadre d'une certification, une revue de conformité du logiciel doit être menée.

h. Le processus de SQA doit produire des documents de suivi de ses activités (sous-section 11.19), y compris des résultats d'audit et la preuve que la revue de conformité de chaque logiciel présenté dans le cadre d'une certification a été réalisée.

## 8.3 REVUE DE CONFORMITE DU LOGICIEL

La revue de conformité du logiciel a pour objectif d'obtenir l'assurance, pour un logiciel présenté dans le cadre d'une certification, que les processus du cycle de vie du logiciel sont achevés, que ses données sont complètes, et que le Code Objet Exécutable est géré en configuration et peut être régénéré.

Cette revue doit établir que :

a. Les activités du cycle de vie du logiciel planifiées en vue d'un crédit de certification, y compris la génération des données du cycle de vie du logiciel, ont été achevées et une trace de cette réalisation a été enregistrée.

b. Les données du cycle de vie du logiciel développées à partir de spécifications particulières du système, des exigences liées à la sécurité ou des spécifications du logiciel, sont traçables vers l'ensemble de ces exigences.

c. Les données du cycle de vie sont conformes aux plans et règles, et sont gérées en configuration en accord avec le Plan de Gestion de Configuration du Logiciel.

d. Les rapports d'anomalie sont conformes au Plan de Gestion de Configuration du Logiciel, ont été évalués, et leur état est enregistré.

e. Les écarts avec les spécifications du logiciel sont enregistrés et approuvés.

f. Le Code Objet Exécutable peut être régénéré à partir du Code Source archivé.

g. Le logiciel approuvé peut être chargé avec succès au moyen des procédures établies.

h. Les rapports d'anomalie en suspens provenant d'une précédente revue de conformité sont réévalués afin de déterminer leur état.

i. Lorsqu'un crédit de certification est demandé pour du logiciel développé antérieurement, il y a traçabilité entre le référentiel du logiciel courant et le référentiel antérieur avec ses modifications approuvées.

*NOTA : Pour les modifications de logiciel postérieures à la certification, on peut réaliser un sous-ensemble des activités de la revue de conformité, en fonction de l'importance des modifications.*

# SECTION 9

# CERTIFICATION LIAISON PROCESS

The objective of the certification liaison process is to establish communication and understanding between the applicant and the certification authority throughout the software life cycle to assist the certification process.

The certification liaison process is applied as defined by the software planning process (section 4) and the Plan for Software Aspects of Certification (subsection 11.1). Table A-10 of Annex A is a summary of the objectives and outputs of this process.

## 9.1 MEANS OF COMPLIANCE AND PLANNING

The applicant proposes a means of compliance that defines how the development of the airborne system or equipment will satisfy the certification basis. The Plan for Software Aspects of Certification (subsection 11.1) defines the software aspects of the airborne system or equipment within the context of the proposed means of compliance. This plan also states the software level(s) as determined by the system safety assessment process. The applicant should:

a. Submit the Plan for Software Aspects of Certification and other requested data to the certification authority for review at a point in time when the effects of changes are minimal, that is, when they can be managed within project constraints.

b. Resolve issues identified by the certification authority concerning the planning for the software aspects of certification.

c. Obtain agreement with the certification authority on the Plan for Software Aspects of Certification.

## 9.2 COMPLIANCE SUBSTANTIATION

The applicant provides evidence that the software life cycle processes satisfy the software plans. Certification authority reviews may take place at the applicant's facilities or the applicant's suppliers' facilities. This may involve discussions with the applicant or its suppliers. The applicant arranges these reviews of the activities of the software life cycle processes and makes software life cycle data available as needed. The applicant should:

a. Resolve issues raised by the certification authority as a result of its reviews.

b. Submit the Software Accomplishment Summary (subsection 11.20) and Software Configuration Index (subsection 11.16) to the certification authority.

c. Submit or make available other data or evidence of compliance requested by the certification authority.

## 9.3 MINIMUM SOFTWARE LIFE CYCLE DATA THAT IS SUBMITTED TO CERTIFICATION AUTHORITY

The minimum software life cycle data that is submitted to the certification authority is:

- Plan for Software Aspects of Certification.
- Software Configuration Index.
- Software Accomplishment Summary.

# SECTION 9

# COORDINATION POUR LA CERTIFICATION

Le processus de coordination pour la certification a pour objectif d'établir la communication et la compréhension entre le postulant et l'Autorité de certification pendant tout le cycle de vie du logiciel afin d'aider au processus de certification.

Le processus de coordination pour la certification s'applique de la manière définie par la planification du logiciel (section 4) et le Plan des Aspects Logiciels de la Certification (sous-section 11.1). Le Tableau A-10 de l'Annexe A constitue un résumé des objectifs et des produits de ce processus.

## 9.1 MOYENS DE CONFORMITE ET PLANIFICATION

Le postulant propose un moyen de conformité définissant de quelle manière le développement du système ou équipement de bord satisfait les bases de la certification. Le Plan des Aspects Logiciels de la Certification (sous-section 11.1) définit les aspects logiciels du système ou équipement de bord dans le contexte du moyen de conformité proposé. Ce plan établit aussi le(s) niveau(x) logiciel(s), déterminé(s) par l'analyse de sécurité du système. Le postulant doit :

a. Soumettre pour revue le Plan des Aspects Logiciels de la Certification et les autres données demandées à l'Autorité de certification à un moment tel que les effets de modifications soient minimaux, c'est-à-dire quand on peut les gérer avec les contraintes du projet.

b. Résoudre les questions identifiées par l'Autorité de certification relativement à la planification des aspects logiciels de la certification.

c. Obtenir l'accord de l'Autorité de certification sur le Plan des Aspects Logiciels de la Certification.

## 9.2 JUSTIFICATION DE CONFORMITE

Le postulant démontre que les cycles de vie du logiciel satisfont les plans du logiciel. Les revues par l'Autorité de certification peuvent se faire dans les locaux du postulant ou dans ceux des fournisseurs du postulant. Cela peut impliquer des discussions avec le postulant ou ses fournisseurs. Le postulant organise ces revues et tient à disposition les données du cycle de vie en cas de nécessité. Le postulant doit :

a. Résoudre les questions soulevées par l'Autorité de certification du fait de ses revues.

b. Soumettre à l'Autorité de certification le Résumé des Travaux Réalisés (sous-section 11.20) et le Répertoire de la Configuration du Logiciel (sous-section 11.16).

c. Soumettre ou tenir à disposition toute autre donnée ou preuve de conformité demandée par l'Autorité de certification.

## 9.3 DONNEES MINIMALES DU CYCLE DE VIE DU LOGICIEL SOUMISES A L'AUTORITE DE CERTIFICATION

Les données minimales du cycle de vie du logiciel soumises à l'Autorité de certification sont les suivantes :

- Plan des Aspects Logiciels de la Certification.
- Répertoire de la Configuration du Logiciel.
- Résumé des Travaux Réalisés.

## 9.4 SOFTWARE LIFE CYCLE DATA RELATED TO TYPE DESIGN

Unless otherwise agreed by the certification authority, the regulations concerning retrieval and approval of software life cycle data related to the type design applies to:

- Software Requirements Data.
- Design Description.
- Source Code.
- Executable Object Code.
- Software Configuration Index.
- Software Accomplishment Summary.

## 9.4 DONNEES DU CYCLE DE VIE DU LOGICIEL LIEES A UNE DEFINITION DE TYPE

Sauf accord de l'Autorité de certification, les règles relatives à la recherche et l'approbation des données du cycle de vie du logiciel liées a une définition de type s'appliquent aux éléments suivants :

- Données de Spécification du Logiciel.
- Description de Conception.
- Code Source.
- Code Objet Exécutable.
- Répertoire de la Configuration du Logiciel.
- Résumé des Travaux Réalisés.

# SECTION 10

# OVERVIEW OF AIRCRAFT AND ENGINE CERTIFICATION

This section is an overview of the certification process for aircraft and engines with respect to software aspects of airborne systems and equipment, and is provided for information purposes only. The certification authority considers the software as part of the airborne system or equipment installed on the aircraft or engine; that is, the certification authority does not approve the software as a unique, stand-alone product.

## 10.1 CERTIFICATION BASIS

The certification authority establishes the certification basis for the aircraft or engine in consultation with the applicant. The certification basis defines the particular regulations together with any special conditions which may supplement the published regulations.

For modified aircraft or engines, the certification authority considers the impact the modification has on the certification basis originally established for the aircraft or engine. In some cases, the certification basis for the modification may not change from the original certification basis; however, the original means of compliance may not be applicable for showing that the modification complies with the certification basis and may need to be changed.

## 10.2 SOFTWARE ASPECTS OF CERTIFICATION

The certification authority assesses the Plan for Software Aspects of Certification for completeness and consistency with the means of compliance that was agreed upon to satisfy the certification basis. The certification authority satisfies itself that the software level(s) proposed by the applicant is consistent with the outputs of the system safety assessment process and other system life cycle data. The certification authority informs the applicant of issues with the proposed software plans that need to be satisfied prior to certification authority agreement.

## 10.3 COMPLIANCE DETERMINATION

Prior to certification, the certification authority determines that the aircraft or engine (including the software aspects of its systems or equipment) complies with the certification basis. For the software, this is accomplished by reviewing the Software Accomplishment Summary and evidence of compliance. The certification authority uses the Software Accomplishment Summary as an overview for the software aspects of certification.

The certification authority may review at its discretion the software life cycle processes and their outputs during the software life cycle as discussed in subsection 9.2.

# SECTION 10

# VUE GENERALE SUR LA CERTIFICATION DES AERONEFS ET DES MOTEURS

Cette section constitue une vue générale de la certification des aéronefs et des moteurs, en ce qui concerne les aspects logiciels des systèmes et équipements de bord, et ne figure ici que dans un but d'information. L'Autorité de certification considère le logiciel comme une partie du système ou équipement de bord installé sur l'aéronef ou le moteur ; ce qui veut dire que l'Autorité de certification n'approuve pas le logiciel comme un produit isolé unique.

## 10.1 BASES DE CERTIFICATION

L'Autorité de certification établit les bases de certification pour l'aéronef ou le moteur en consultation avec le postulant. Les bases de certification définissent les règles particulières en même temps que toute condition spéciale qui pourrait s'ajouter aux règles publiées.

Pour les aéronefs ou moteurs modifiés, l'Autorité de certification prend en considération l'impact de la modification sur les bases de certification établies à l'origine pour l'aéronef ou le moteur. Dans certains cas, les bases de certification pour la modification peuvent ne pas changer les bases de certification d'origine ; toutefois, les moyens de conformité d'origine peuvent ne pas être applicables à la démonstration de la conformité de la modification vis à vis des bases de certification et peuvent devoir être modifiés.

## 10.2 ASPECTS LOGICIELS DE LA CERTIFICATION

L'Autorité de certification évalue le Plan des Aspects Logiciels de la Certification du point de vue de son exhaustivité et de sa cohérence avec les moyens de conformité admis pour la satisfaction des bases de certification. L'Autorité de certification est satisfaite si le(s) niveau(x) logiciel(s) proposé(s) sont cohérents avec les sorties de l'analyse de sécurité du système et les autres données du cycle de vie du système. L'Autorité de certification informe le postulant des questions soulevées dans les plans de logiciel proposés, et qui doivent être résolues préalablement à l'accord de l'Autorité de certification.

## 10.3 ETABLISSEMENT DE LA CONFORMITE

Préalablement à la certification, l'Autorité de certification établit que l'aéronef ou le moteur (y compris les aspects logiciels de ce système ou équipement) est conforme aux bases de certification. Pour le logiciel, ceci se fait par revue du Résumé des Travaux Réalisés et démonstration de la conformité. L'Autorité de certification utilise le Résumé des Travaux Réalisés comme une vue générale des aspects logiciels de la certification.

L'Autorité de certification peut revoir à sa discrétion les processus du cycle de vie du logiciel et leurs produits au cours de son cycle de vie, comme cela est traité dans la sous-section 9.2.

# SECTION 11

# SOFTWARE LIFE CYCLE DATA

Data is produced during the software life cycle to plan, direct, explain, define, record, or provide evidence of activities. This data enables the software life cycle processes, system or equipment certification, and post-certification modification of the software product. This section discusses the characteristics, form, configuration management controls, and content of the software life cycle data.

The characteristics of the software life cycle data are:

a. Unambiguous: Information is unambiguous if it is written in terms which only allow a single interpretation, aided if necessary by a definition.

b. Complete: Information is complete when it includes necessary, relevant requirements and/or descriptive material, responses are defined for the range of valid input data, figures used are labeled, and terms and units of measure are defined.

c. Verifiable: Information is verifiable if it can be checked for correctness by a person or tool.

d. Consistent: Information is consistent if there are no conflicts within it.

e. Modifiable: Information is modifiable if it is structured and has a style such that changes can be made completely, consistently, and correctly while retaining the structure.

f. Traceable: Information is traceable if the origin of its components can be determined.

In addition, this guidance applies:

g. Form: The form should provide for the efficient retrieval and review of software life cycle data throughout the service life of the airborne system or equipment. The data and the specific form of the data should be specified in the Plan for Software Aspects of Certification.

*NOTE 1: The software life cycle data may take a number of forms. For example, it can be in written form, as computer files stored on magnetic media, or as displays on a remote terminal. It may be packaged as individual documents, combined into larger documents, or distributed across several documents.*

*NOTE 2: The Plan for Software Aspects of Certification and the Software Accomplishment Summary may be required as separate printed documents by some certification authorities.*

Software life cycle data can be placed in one of two categories associated with the software configuration management controls applied: Control Category 1 (CC1) and Control Category 2 (CC2) (subsection 7.3). This distinction provides a means to manage development costs where less stringent controls can be applied without a reduction in safety. The minimum control category assigned to each data item, and its variation by software level is specified in Annex A.

The following data descriptions define the data generally produced to aid in the software aspects of the certification process. The descriptions are not intended to describe all data necessary to develop a software product, and are not intended to imply a particular data packaging method or organization of the data within a package.

In addition to the data defined in these subsections, other data may be produced as evidence to aid the certification process.

# SECTION 11

## DONNEES DU CYCLE DE VIE DU LOGICIEL

Des données sont générées au cours du cycle de vie pour planifier, orienter, expliquer, définir, enregistrer, ou apporter des preuves relatives à des activités. Ces données rendent possibles les processus du cycle de vie du logiciel, la certification du système ou de l'équipement et les modifications du logiciel postérieures à la certification. Cette section traite des caractéristiques, de la forme, des règles de gestion de configuration et du contenu des données du cycle de vie.

Les caractéristiques des données du cycle de vie du logiciel sont d'être :

a. Non-ambiguës : Les informations sont non ambiguës si elles sont écrites en termes qui ne permettent qu'une seule interprétation, assistée au besoin par une définition.

b. Complètes : Les informations sont complètes quand elles comportent des exigences et/ou des matériaux descriptifs nécessaires et pertinents, quand les réponses sont définies pour tout le domaine d'entrées valides, quand les valeurs numériques employées sont caractérisées, les termes et unités de mesure définis.

c. Vérifiables : Les informations sont vérifiables si on peut contrôler qu'elles sont correctes au moyen d'une personne ou d'un outil.

d. Cohérentes : Les informations sont cohérentes si elles ne renferment pas de contradiction.

e. Modifiables : Les informations sont modifiables si leur structure et leur présentation permettent de faire des modifications de manière complète, cohérente et correcte, tout en maintenant la structure.

f. Traçables : Les informations sont traçables si on peut déterminer l'origine de leurs éléments.

En outre, les directives suivantes s'appliquent :

g. Forme : La forme doit permettre une recherche et une revue efficaces des données du cycle de vie du logiciel pendant toute la durée d'utilisation du système ou équipement de bord. Les données et leur forme précise doivent être spécifiées dans le Plan des Aspects Logiciels de la Certification.

*NOTA 1 : Les données du cycle de vie d'un logiciel peuvent prendre un certain nombre de formes. Elles peuvent être accessibles par exemple sous forme écrite, sur support magnétique, sous forme de fichiers informatiques, ou visualisées sur un terminal déporté. Elles peuvent être exploitées en tant que documents isolés, combinées avec des documents plus vastes, ou réparties sur* plusieurs documents.

*NOTA 2 : Le Plan des Aspects Logiciels de la Certification et le Résumé des Travaux Réalisés peuvent être exigés par certaines Autorités de certification sous la forme de documents imprimés séparés.*

Les données du cycle de vie du logiciel peuvent être classées dans l'une des deux catégories associées aux règles de gestion de configuration du logiciel : la Catégorie de Contrôle 1 (CC1) et la Catégorie de Contrôle 2 (CC2) (sous-section 7.3). Cette distinction fournit un moyen de gérer les coûts de développement là où des contrôles moins sévères peuvent être employés sans réduction de sécurité. La catégorie minimale de contrôle attribuée à chaque unité d'information avec la variation par niveau logiciel est spécifiée dans l'Annexe A.

Les descriptions de données qui suivent définissent les données généralement produites pour aider aux aspects logiciels de la certification. Les descriptions n'ont pas pour but de décrire toutes les données nécessaires au développement d'un produit logiciel, ni d'impliquer une méthode particulière de conditionnement, d'organisation et de structuration de ces données.

Outre les données définies dans ces sous-sections, d'autres données peuvent être produites à titre de preuve pour aider à la certification.

h. Control: If intended to be used for this purpose, this data should be defined in the software plan which defines the process for which the data will be produced. While the nature and content of this data may vary, as a minimum, CC2 controls should be applied. Examples include memoranda and meeting minutes.

## 11.1 PLAN FOR SOFTWARE ASPECTS OF CERTIFICATION

The Plan for Software Aspects of Certification is the primary means used by the certification authority for determining whether an applicant is proposing a software life cycle that is commensurate with the rigor required for the level of software being developed. This plan should include:

a. System overview: This section provides an overview of the system, including a description of its functions and their allocation to the hardware and software, the architecture, processor(s) used, hardware/software interfaces, and safety

b. Software overview: This section briefly describes the software functions with emphasis on the proposed safety and partitioning concepts, for example, resource sharing, redundancy, multiple-version dissimilar software, fault tolerance, and timing and scheduling strategies.

c. Certification considerations: This section provides a summary of the certification basis, including the means of compliance, as relating to the software aspects of certification. This section also states the proposed software level(s) and summarizes the justification provided by the system safety assessment process, including potential software contributions to failure conditions.

d. Software life cycle: This section defines the software life cycle to be used and includes a summary of each software life cycle and its processes for which detailed information is defined in their respective software plans. The summary explains how the objectives of each software life cycle process will be satisfied, and specifies the organizations to be involved, the organizational responsibilities, and the system life cycle processes and certification liaison process responsibilities.

e. Software life cycle data: This section specifies the software life cycle data that will be produced and controlled by the software life cycle processes. This section also describes the relationship of the data to each other or to other data defining the system, the software life cycle data to be submitted to the certification authority, the form of the data, and the means by which software life cycle data will be made available to the certification authority.

f. Schedule: This section describes the means the applicant will use to provide the certification authority with visibility of the activities of the software life cycle processes so reviews can be planned.

g. Additional considerations: This section describes specific features that may affect the certification process, for example, alternative methods of compliance, tool qualification, previously developed software, option-selectable software, user-modifiable software, COTS software, field-loadable software, multiple-version dissimilar software, and product service history.

## 11.2 SOFTWARE DEVELOPMENT PLAN

The Software Development Plan includes the objectives, standards and software life cycle(s) to be used in the software development processes. It may be included in the Plan for Software Aspects of Certification. This plan should include:

h. Contrôle : Si leur utilisation est prévue dans ce but, ces données doivent être définies dans le plan qui définit le processus au cours duquel ces données sont produites. Même si la nature et le contenu de ces données peuvent varier, on appliquera au minimum les contrôles CC2. Parmi les exemples on peut citer les notes et les comptes rendus de réunions.

## 11.1 PLAN DES ASPECTS LOGICIELS DE LA CERTIFICATION

Le Plan des Aspects Logiciels de la Certification est le support initial utilisé par l'Autorité de certification pour déterminer si un postulant propose un cycle de vie du logiciel en accord avec le niveau logiciel à développer. Ce plan doit comprendre les éléments suivants :

a. Vue d'ensemble du système : Cette section donne une vue d'ensemble du système, y compris une description de ses fonctions et de leur allocation au matériel et au logiciel, l'architecture, le(s) processeur(s) utilisé(s), les interfaces matériel/logiciel, et les caractéristiques relatives à la sécurité.

b. Vue d'ensemble du logiciel : Cette section décrit brièvement les fonctions du logiciel en insistant sur les concepts proposés pour la sécurité et le partitionnement, par exemple le partage des ressources, la redondance, les logiciels à versions multiples dissimilaires, la tolérance aux fautes, et les stratégies de séquencement et d'ordonnancement.

c. Considérations relatives à la certification : Cette section fournit un résumé de la base de certification, comprenant les moyens de conformité liés aux aspects logiciels de la certification. Cette section énonce aussi le(s) niveau(x) logiciel(s) proposé(s) et résume la justification apportée par l'analyse de sécurité du système, y compris les contributions potentielles du logiciel aux conditions de panne.

d. Cycle de vie du logiciel : Cette section définit le cycle de vie du logiciel à utiliser et comprend un résumé de chaque cycle de vie et de ses processus. Pour ces derniers, des informations détaillées sont définies dans leurs plans. Le résumé explique de quelle manière les objectifs de chaque cycle de vie seront satisfaits. Il mentionne les organisations impliquées et les responsabilités au sein de chaque organisation. Il mentionne enfin les responsabilités des processus du cycle de vie du système et de celui de la coordination pour la certification.

e. Données du cycle de vie du logiciel : Cette section précise les données du cycle de vie du logiciel qui seront produites et contrôlées par les processus de ce cycle de vie. Cette section décrit aussi les liens entre données ou avec celles définissant le système, les données du cycle de vie du logiciel à soumettre à l'Autorité de certification, la forme de ces données, et le moyen par lequel elles seront mises à sa disposition.

f. Planning : Cette section décrit le moyen que le postulant utilisera pour donner visibilité à l'Autorité de certification sur les activités du cycle de vie du logiciel afin de permettre la planification de revues.

g. Considérations complémentaires (section 12) : Cette section décrit des caractéristiques spécifiques pouvant affecter la certification, par exemple l'utilisation de méthodes de substitution, la nécessité d'une qualification d'outil, l'utilisation de logiciels développés antérieurement, de logiciels à options, de logiciels modifiables par l'utilisateur, de logiciels du commerce sur étagère, de logiciels téléchargeables, de logiciels à versions multiples dissimilaires ou de l'historique du produit en service.

## 11.2 PLAN DE DEVELOPPEMENT DU LOGICIEL

Le Plan de Développement du Logiciel comprend les objectifs, les règles, et le(s) cycle(s) de vie du logiciel à utiliser dans les processus de développement du logiciel. Ce plan peut être inclus dans le Plan des Aspects Logiciels de la Certification. Ce plan doit comporter les éléments suivants :

a. Standards: Identification of the Software Requirements Standards, Software Design Standards and Software Code Standards for the project. Also, references to the standards for previously developed software, including COTS software, if those standards are different.

b. Software life cycle: A description of the software life cycle processes to be used to form the specific software life cycle(s) to be used on the project, including the transition criteria for the software development processes. This description is distinct from the summary provided in the Plan for Software Aspects of Certification, in that it provides the detail necessary to ensure proper implementation of the software life cycle processes.

c. Software development environment: A statement of the chosen software development environment in terms of hardware and software, including:

   (1) The chosen requirements development method(s) and tools to be used.

   (2) The chosen design method(s) and tools to be used.

   (3) The programming language(s), coding tools, compilers, linkage editors and loaders to be used.

   (4) The hardware platforms for the tools to be used.

## 11.3 SOFTWARE VERIFICATION PLAN

The Software Verification Plan is a description of the verification procedures to satisfy the software verification process objectives. These procedures may vary by software level as defined in the tables of Annex A. This plan should include:

a. Organization: Organizational responsibilities within the software verification process and interfaces with the other software life cycle processes.

b. Independence: A description of the methods for establishing verification independence, when required.

c. Verification methods: A description of the verification methods to be used for each activity of the software verification process.

   (1) Review methods, including checklists or other aids.

   (2) Analysis methods, including traceability and coverage

   (3) Testing methods, including guidelines that establish the test case selection process, the test procedures to be used, and the test data to be produced.

d. Verification environment: A description of the equipment for testing, the testing and analysis tools, and the guidelines for applying these tools and hardware test equipment (see also paragraph 4.4.3, item b for guidance on indicating target computer and simulator or emulator differences

e. Transition criteria: The transition criteria for entering the software verification process defined in this plan.

f. Partitioning considerations: If partitioning is used, the methods used to verify the integrity of the partitioning.

g. Compiler assumptions: A description of the assumptions made by the applicant about the correctness of the compiler, linkage editor or loader (paragraph 4.4.2).

h. Reverification guidelines: For software modification, a description of the methods for identifying the affected areas of the software and the changed parts of the Executable Object Code. The reverification should ensure that previously reported errors or classes of errors have been eliminated.

a. Règles : Identification des Règles de Spécification du Logiciel, des Règles de Conception du Logiciel, et des Règles de Codage pour le projet. Egalement, les références aux règles pour les logiciels développés antérieurement, y compris les logiciels du commerce sur étagère, si ces règles sont différentes.

b. Cycle du vie du logiciel : Description des cycles de vie du logiciel à utiliser pour constituer le(s) cycle(s) de vie spécifique(s) du logiciel à employer dans le projet, y compris les critères de transition pour les processus de développement du logiciel. Cette description est différente du résumé fourni dans le Plan des Aspects Logiciels de la Certification, en ce sens qu'elle donne les détails nécessaires à la mise en oeuvre adéquate des processus du cycle de vie du logiciel.

c. Environnement de développement du logiciel : Une définition de l'environnement choisi pour le développement du logiciel, en termes de matériel et de logiciel, comprenant :

   (1) Les méthodes choisies pour le développement des spécifications et les outils qui seront utilisés.

   (2) Le(s) méthode(s) de conception choisies et les outils qui seront utilisés.

   (3) Le(s) langage(s) de programmation, les outils de codage, les compilateurs, les éditeurs de liens et les chargeurs qui seront utilisés.

   (4) Les environnements matériels pour les outils qui seront utilisés.

## 11.3 PLAN DE VERIFICATION DU LOGICIEL

Le Plan de Vérification du Logiciel est une description des procédures de vérification pour satisfaire les objectifs du processus de vérification du logiciel. Ces procédures peuvent varier selon le niveau logiciel de la manière définie par les tableaux de l'Annexe A. Ce plan doit comporter les éléments suivants :

a. Organisation : Organisation et responsabilités au sein du processus de vérification du logiciel et interfaces avec les autres processus du cycle de vie du logiciel.

b. Indépendance : Description des méthodes pour établir l'indépendance de la vérification, si exigé.

c. Méthodes de vérification : Description des méthodes de vérification à utiliser pour chaque activité du processus de vérification du logiciel.

   (1) Méthodes de revue, y compris les listes de contrôle (check list) ou autres aides.

   (2) Méthodes d'analyse, y compris l'analyse de traçabilité et de couverture.

   (3) Méthodes de test, y compris les recommandations pour le choix des jeux de test, pour les procédures de test à utiliser, et pour les données de test à produire.

d. Environnement de vérification : Description de l'équipement de test, des outils de test et d'analyse, et des recommandations pour l'utilisation de ces outils et équipement de test (voir aussi paragraphe 4.4.3 b, en ce qui concerne les directives relatives aux différences entre machine cible et simulateur ou émulateur).

e. Critères de transition : Critères de transition utilisés pour engager le processus de vérification du logiciel défini dans ce plan.

f. Considérations relatives au partitionnement : Méthodes utilisées pour vérifier l'intégrité du partitionnement s'il est employé.

g. Hypothèses relatives au compilateur : Description des hypothèses faites par le postulant relatives à la correction du compilateur, de l'éditeur de liens, ou du chargeur (paragraphe 4.4.2).

h. Directives pour la revérification : Pour les modifications de logiciel, description des méthodes d'identification des parties du logiciel concernées et des éléments modifiés dans le Code Objet Exécutable. La revérification doit garantir que les erreurs ou classes d'erreurs découvertes antérieurement ont été éliminées.

i. Previously developed software: For previously developed software, if the initial compliance baseline for the verification process does not comply with this document, a description of the methods to satisfy the objectives of this document.

j. Multiple-version dissimilar software: If multiple-version dissimilar software is used, a description of the software verification process activities (paragraph 12.3.3).

## 11.4 SOFTWARE CONFIGURATION MANAGEMENT PLAN

The Software Configuration Management Plan establishes the methods to be used to achieve the objectives of the software configuration management (SCM) process throughout the software life cycle. This plan should include:

a. Environment: A description of the SCM environment to be used, including procedures, tools, methods, standards, organizational responsibilities, and interfaces.

b. Activities: A description of the SCM process activities in the software life cycle that will satisfy the objectives for:

(1) Configuration identification: Items to be identified, when they will be identified, the identification methods for software life cycle data (for example, part numbering), and the relationship of software identification and airborne system or equipment identification.

(2) Baselines and traceability: The means of establishing baselines, what baselines will be established, when these baselines will be established, the software library controls, and the configuration item and baseline traceability.

(3) Problem reporting: The content and identification of problem reports for the software product and software life cycle processes, when they will be written, the method of closing problem reports, and the relationship to the change control

(4) Change control: Configuration items and baselines to be controlled, when they will be controlled, the problem/change control activities that control them, pre-certification controls, post-certification controls, and the means of preserving the integrity of baselines and configuration items.

(5) Change review: The method of handling feedback from and to the software life cycle processes; the methods of assessing and prioritizing problems, approving changes, and handling their resolution or change implementation; and the relationship of these methods to the problem reporting and change control activities.

(6) Configuration status accounting: The data to be recorded to enable reporting configuration management status, definition of where that data will be kept, how it will be retrieved for reporting, and when it will be available.

(7) Archive, retrieval, and release: The integrity controls, the release method and authority, and data retention.

(8) Software load control: A description of the software load control safeguards and records.

(9) Software life cycle environment controls: Controls for the tools used to develop, build, verify and load the software, addressing items 1 through 7 above. This includes control of tools to be

(10) Software life cycle data controls: Controls associated with Control Category 1 and Control Category 2 data.

i. Logiciels développés antérieurement : Pour les logiciels développés antérieurement, si le référentiel initial pour le processus de vérification n'est pas conforme à ce document, description des méthodes destinées à la satisfaction des objectifs de ce document.

j. Logiciels à versions multiples dissimilaires : Si on utilise un logiciel à versions multiples dissimilaires, description des activités du processus de vérification du logiciel (paragraphe 12.3.3).

## 11.4 PLAN DE GESTION DE CONFIGURATION DU LOGICIEL

Le Plan de Gestion de Configuration du Logiciel détermine les méthodes à utiliser pour atteindre les objectifs de gestion de configuration du logiciel (SCM) pendant tout le cycle de vie du logiciel. Ce plan doit comporter les éléments suivants :

a. Environnement : Description de l'environnement de SCM, y compris les procédures, outils, méthodes, règles, l'organisation et les responsabilités ainsi que les interfaces.

b. Activités : Description des activités du processus de SCM qui permettent de satisfaire les objectifs relatifs aux points suivants :

(1) Identification de la configuration : Eléments à identifier, moment où ils sont identifiés, méthodes d'identification des données du cycle de vie (par exemple, constitution des identificateurs), et relation entre les règles d'identification du logiciel et celles du système ou équipement de bord.

(2) Référentiels et traçabilité : Moyens utilisés pour établir des référentiels, identification des référentiels à établir, moment où ils seront établis, procédures de gestion de la "bibliothèque du logiciel" (cf glossaire), et traçabilité entre éléments de configuration et référentiels.

(3) Compte rendu des anomalies : Contenu et identification des rapports d'anomalies sur le produit logiciel et les processus du cycle de vie, moment où ils seront rédigés, méthode de clôture des rapports d'anomalies, et relation avec l'activité de gestion des modifications.

(4) Gestion des modifications : Eléments de configuration et référentiels à contrôler, moment où ils sont contrôlés, activités de gestion des anomalies/modifications qui assurent leur maîtrise, contrôles avant certification, contrôles après certification, et moyens pour préserver l'intégrité des référentiels et des éléments de configuration.

(5) Revue des modifications : Méthode de prise en compte des retours d'informations en provenance et en direction des processus du cycle de vie du logiciel ; méthodes pour évaluer les anomalies et les classer par priorité, pour approuver les modifications, prendre respectivement en charge leur résolution et leur réalisation ; et relation entre ces méthodes et les activités de compte rendu des anomalies et de gestion des modifications.

(6) Suivi des états de configuration : Définition des données à enregistrer afin de pouvoir rendre compte des états de gestion de configuration, de la définition des lieux de stockage de ces données, des modalités de recherche à des fins de compte rendu, et de leur date de disponibilité.

(7) Archivage, restauration, et mise à disposition : Contrôles d'intégrité, méthode et responsabilité pour la mise à disposition, et règles de conservation des données.

(8) Maîtrise du chargement du logiciel : Description des protections pour maîtriser le chargement du logiciel et des enregistrements associés.

(9) Maîtrise de l'environnement de développement du logiciel : Règles de gestion des outils utilisés pour développer, produire, vérifier, et charger le logiciel, prenant en compte les points 1 à 7 ci-dessus. Ceci s'applique aussi aux outils à qualifier.

(10) Gestion des données du cycle de vie du logiciel : Règles de contrôle associées aux données de Catégorie de Contrôle 1 et de Catégorie de Contrôle 2.

c. Transition criteria: The transition criteria for entering the SCM process.

d. SCM data: A definition of the software life cycle data produced by the SCM process, including SCM Records, the Software Configuration Index and the Software Life Cycle Environment Configuration Index.

e. Supplier control: The means of applying SCM process requirements to sub-tier suppliers.

## 11.5 SOFTWARE QUALITY ASSURANCE PLAN

The Software Quality Assurance Plan establishes the methods to be used to achieve the objectives of the software quality assurance (SQA) process. The SQA Plan may include descriptions of process improvement, metrics, and progressive management methods. This plan should include:

a. Environment: A description of the SQA environment, including scope, organizational responsibilities and interfaces, standards, procedures, tools and methods.

b. Authority: A statement of the SQA authority, responsibility, and independence, including the approval authority for software products.

c. Activities: The SQA activities that are to be performed for each software life cycle process and throughout the software life cycle including:

(1) SQA methods, for example, reviews, audits, reporting, inspections, and monitoring of the software life cycle processes.

(2) Activities related to the problem reporting, tracking and corrective action system.

(3) A description of the software conformity review activity.

a. Transition criteria: The transition criteria for entering the SQA process.

b. Timing: The timing of the SQA process activities in relation to the activities of the software life cycle processes.

c. SQA Records: A definition of the records to be produced by the SQA process.

d. Supplier control: A description of the means of ensuring that sub-tier suppliers' processes and outputs will comply with the SQA Plan.

## 11.6 SOFTWARE REQUIREMENTS STANDARDS

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

a. The methods to be used for developing software requirements, such as structured methods.

b. Notations to be used to express requirements, such as data flow diagrams and formal specification languages.

c. Constraints on the use of the requirement development tools.

d. The method to be used to provide derived requirements to the system process.

c. Critères de transition : Critères de transition utilisés pour engager le processus de SCM.

d. Données de gestion de configuration : Définition des données du cycle de vie du logiciel générées par le processus de SCM, y compris les Documents SCM, le Répertoire de la Configuration du Logiciel, et le Répertoire de la Configuration de l'Environnement du Logiciel.

e. Maîtrise des fournisseurs : Moyens de répercuter sur les fournisseurs les exigences du processus de SCM.

## 11.5 PLAN D'ASSURANCE QUALITE DU LOGICIEL

Le Plan d'Assurance Qualité du Logiciel définit les méthodes à utiliser pour atteindre les objectifs du processus d'assurance qualité du logiciel (SQA). Le plan SQA peut comprendre des descriptions d'amélioration de ce processus, des métriques, et des méthodes de gestion continue de la qualité. Ce plan doit comporter les éléments suivants :

a. Environnement : Description de l'environnement de SQA, y compris le champ d'application, l'organisation et les responsabilités ainsi que les interfaces, règles, procédures, outils, et méthodes.

b. Autorité : Déclaration établissant l'autorité, la responsabilité, et l'indépendance de la fonction SQA, y compris l'autorité d'approbation des produits logiciels.

c. Activités : Activités de SQA à réaliser pour chaque processus du cycle de vie du logiciel et au long du cycle de vie, y compris les éléments ci-dessous :

   (1) Méthodes de SQA, par exemple revues, audits, comptes rendus, inspections, et surveillance des processus du cycle de vie.

   (2) Activités liées au système de compte rendu et de suivi des anomalies, et d'actions correctives.

   (3) Description de l'activité de revue de conformité du logiciel.

d. Critères de transition : Critères de transition utilisés pour engager le processus de SQA.

e. Séquencement : Séquencement des activités du processus de SQA en relation avec les activités du cycle de vie du logiciel.

f. Documents SQA : Définition des documents que le processus de SQA doit générer.

g. Maîtrise des fournisseurs : Description des moyens garantissant que les processus et les produits des fournisseurs sont conformes au Plan SQA.

## 11.6 REGLES DE SPECIFICATION DU LOGICIEL

Les Règles de Spécification du Logiciel ont pour but de définir les méthodes, règles, et outils à utiliser pour le développement des exigences de haut niveau. Ces règles doivent comprendre :

a. Les méthodes à employer pour le développement des spécifications du logiciel, telles que les méthodes d'analyse structurées.

b. Le formalisme à utiliser pour exprimer les exigences, tel que diagrammes de flux de données et langages de spécification formels.

c. Les contraintes d'utilisation des outils de spécification.

d. La méthode à utiliser pour fournir au processus du système les exigences dérivées.

## 11.7 SOFTWARE DESIGN STANDARDS

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

a. Design description method(s) to be used.

b. Naming conventions to be used.

c. Conditions imposed on permitted design methods, for example, scheduling, and the use of interrupts and event-driven architectures, dynamic tasking, re-entry, global data, and exception handling, and rationale for their use.

d. Constraints on the use of the design tools.

e. Constraints on design, for example, exclusion of recursion, dynamic objects, data aliases, and compacted expressions.

f. Complexity restrictions, for example, maximum level of nested calls or conditional structures, use of unconditional branches, and number of entry/exit points of code components.

## 11.8 SOFTWARE CODE STANDARDS

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

a. Programming language(s) to be used and/or defined subset(s). For a programming language, reference the data that unambiguously defines the syntax, the control behavior, the data behavior and side-effects of the language. This may require limiting the use of some features of a

b. Source Code presentation standards, for example, line length restriction, indentation, and blank line usage and Source Code documentation standards, for example, name of author, revision history, inputs and outputs, and affected global data.

c. Naming conventions for components, subprograms, variables, and constants.

d. Conditions and constraints imposed on permitted coding conventions, such as the degree of coupling between software components and the complexity of logical or numerical expressions and rationale for their use.

e. Constraints on the use of the coding tools.

## 11.9 SOFTWARE REQUIREMENTS DATA

Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include:

a. Description of the allocation of system requirements to software, with attention to safety-related requirements and potential failure conditions.

b. Functional and operational requirements under each mode of operation.

c. Performance criteria, for example, precision and accuracy.

d. Timing requirements and constraints.

e. Memory size constraints.

## 11.7 REGLES DE CONCEPTION DU LOGICIEL

Les Règles de Conception du Logiciel ont pour but de définir les méthodes, règles, et outils à utiliser pour le développement de l'architecture du logiciel et des exigences de bas niveau. Ces règles doivent comporter :

a. La (ou les) méthode(s) de description de la conception à utiliser.

b. Les conventions de dénomination à utiliser.

c. Les restrictions imposées aux méthodes de conception autorisées, relatives par exemple à l'ordonnancement, à l'utilisation d'architectures pilotées par interruptions ou événements, à l'allocation dynamique des tâches, à la réentrance, à l'utilisation de données globales, au traitement des exceptions, ainsi que la justification de leur utilisation.

d. Les contraintes d'utilisation des outils de conception.

e. Les contraintes de conception, par exemple l'interdiction de la récursivité, des objets dynamiques, des données surnommées (alias), et des expressions compactées.

f. Les restrictions de complexité, relatives par exemple au niveau maximal d'imbrication des appels ou des structures conditionnelles, à l'utilisation de branchements inconditionnels, et au nombre de points d'entrée/sortie des composants de code.

## 11.8 REGLES DE CODAGE DU LOGICIEL

Les Règles de Codage du Logiciel ont pour but de définir les langages de programmation, méthodes, règles, et outils à utiliser pour le codage du logiciel. Ces règles doivent comporter :

a. Le(s) langage(s) de programmation à utiliser et/ou le(s) sous-ensemble(s) défini(s). Pour un langage de programmation, référencer les informations qui définissent de manière non ambiguë la syntaxe, les structures de contrôle ou de données offertes et leur mise en oeuvre ainsi que les effets de bord du langage. Ceci peut imposer de limiter l'utilisation de certaines caractéristiques d'un langage.

b. Les règles de présentation du Code Source, par exemple la longueur limite des lignes, l'indentation, l'utilisation de lignes blanches et les règles de documentation du Code Source, par exemple le nom de l'auteur, l'historique des révisions, les données d'entrée et de sortie, et les données globales modifiées.

c. Les conventions de dénomination des composants, sous-programmes, variables, et constantes.

d. Les conditions et contraintes imposées dans les conventions de codage autorisées, telles que le degré de couplage entre les composants logiciel et la complexité des expressions logiques et numériques, ainsi que la justification de leur utilisation.

e. Les contraintes d'utilisation des outils de codage.

## 11.9 DONNEES DE SPECIFICATION DU LOGICIEL

Les Données de Spécification du Logiciel constituent une définition des exigences de haut niveau, y compris les exigences dérivées. Ces données doivent comprendre :

a. La description de l'allocation des spécifications du système au logiciel, avec une attention particulière pour les exigences liées à la sécurité et les conditions de panne potentielles.

b. Les exigences fonctionnelles et opérationnelles dans chaque mode de fonctionnement.

c. Les critères de performance, par exemple précision et exactitude.

d. Les exigences et contraintes de temps.

e. Les contraintes de taille de mémoire.

f. Hardware and software interfaces, for example, protocols, formats, frequency of inputs and frequency of outputs.

g. Failure detection and safety monitoring requirements.

h. Partitioning requirements allocated to software, how the partitioned software components interact with each other, and the software level(s) of each partition.

## 11.10 DESIGN DESCRIPTION

The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the software high-level requirements. This data should include:

a. A detailed description of how the software satisfies the specified software high-level requirements, including algorithms, data structures, and how software requirements are allocated to processors and tasks.

b. The description of the software architecture defining the software structure to implement the requirements.

c. The input/output description, for example, a data dictionary, both internally and externally throughout the software

d. The data flow and control flow of the design.

e. Resource limitations, the strategy for managing each resource and its limitations, the margins, and the method for measuring those margins, for example, timing and memory.

f. Scheduling procedures and inter-processor/inter-task communication mechanisms, including time-rigid sequencing, preemptive scheduling, Ada rendezvous, and interrupts.

g. Design methods and details for their implementation, for example, software data loading, user-modifiable software, or multiple-version dissimilar software.

h. Partitioning methods and means of preventing partition

i. Descriptions of the software components, whether they are new or previously developed, and, if previously developed, reference to the baseline from which they were taken.

j. Derived requirements resulting from the software design process.

k. If the system contains deactivated code, a description of the means to ensure that the code cannot be enabled in the target computer.

l. Rationale for those design decisions that are traceable to safety-related system requirements.

## 11.11 SOURCE CODE

This data consists of code written in source language(s) and the compiler instructions for generating the object code from the Source Code, and linking and loading data. This data should include the software identification, including the name and date of revision and/or version, as applicable.

## 11.12 EXECUTABLE OBJECT CODE

The Executable Object Code consists of a form of Source Code that is directly usable by the central processing unit of the target computer and is, therefore, the software that is loaded into the hardware or system.

f. Les interfaces entre matériel et logiciel, par exemple protocoles, formats, fréquence des entrées et des sorties.

g. Les exigences relatives à la détection des pannes et aux surveillances.

h. Les exigences allouées au logiciel relatives au partitionnement, telles que le mode d'interaction entre éléments de la partition et le(s) niveau(x) logiciel(s) de chaque élément.

## 11.10 DESCRIPTION DE CONCEPTION

La Description de Conception constitue une définition de l'architecture du logiciel et des exigences de bas niveau qui satisferont les exigences de haut niveau du logiciel. Ces données doivent comprendre :

a. Une description détaillée expliquant comment le logiciel satisfait les exigences de haut niveau spécifiées, y compris les algorithmes, les structures de données, et la manière dont les exigences sont allouées aux processeurs et aux tâches.

b. La description de l'architecture du logiciel définissant la structure retenue pour réaliser les exigences.

c. La description des entrées/sorties, par exemple sous forme d'un dictionnaire des données internes et externes, pour l'ensemble de l'architecture du logiciel.

d. Les flux de données et les flux de contrôle.

e. Les limitations des ressources, la stratégie de gestion de chaque ressource et de ses limitations, les marges et les méthodes de mesure de ces marges, par exemple en ce qui concerne les temps d'exécution et l'occupation mémoire.

f. Les procédures d'ordonnancement et les mécanismes de communication inter-processeurs/inter-tâches, y compris le séquencement à échéances fixes, l'ordonnancement avec préemption, les rendez-vous ADA, et les interruptions.

g. Les méthodes de conception et les détails de leur application, concernant par exemple le téléchargement de données, les logiciels modifiables par l'utilisateur, et les logiciels à versions multiples dissimilaires.

h. Les méthodes de partitionnement et les moyens d'empêcher les violations de partition.

i. La description de chaque composant logiciel, qu'il ait été développé antérieurement ou non, et dans le premier cas le référentiel dont il est issu.

j. Les exigences dérivées résultant du processus de conception du logiciel.

k. Si le système comprend du code désactivé, une description des moyens de garantir que le code ne peut pas être activé sur la machine cible.

l. Justification des décisions de conception qui sont traçables vers les spécifications du système liées à la sécurité.

## 11.11 CODE SOURCE

Ces données sont constituées par le code écrit dans le(s) langage(s) de programmation, les instructions de compilation pour générer le code objet à partir du Code Source, effectuer l'édition de liens et charger le logiciel. Ces données doivent comprendre l'identification du logiciel, y compris le nom et la date de la révision et/ou version, selon le cas.

## 11.12 CODE OBJET EXECUTABLE

Le Code Objet Exécutable est constitué par une forme du Code Source directement utilisable par l'unité centrale de traitement de la machine cible et qui constitue de ce fait, le logiciel qui est chargé dans le matériel ou dans le système.

## 11.13 SOFTWARE VERIFICATION CASES AND PROCEDURES

Software Verification Cases and Procedures detail how the software verification process activities are implemented. This data should include descriptions of the:

a. Review and analysis procedures: Details, supplementary to the description in the Software Verification Plan, which describes the scope and depth of the review or analysis methods to be used.

b. Test cases: The purpose of each test case, set of inputs, conditions, expected results to achieve the required coverage criteria, and the pass/fail criteria.

c. Test procedures: The step-by-step instructions for how each test case is to be set up and executed, how the test results are evaluated, and the test environment to be used.

## 11.14 SOFTWARE VERIFICATION RESULTS

The Software Verification Results are produced by the software verification process activities. Software Verification Results should:

a. For each review, analysis and test, indicate each procedure that passed or failed during the activities and the final pass/fail results

b. Identify the configuration item or software version reviewed, analyzed or tested.

c. Include the results of tests, reviews and analyses, including coverage analyses and traceability analyses.

## 11.15 SOFTWARE LIFE CYCLE ENVIRONMENT CONFIGURATION INDEX

The Software Life Cycle Environment Configuration Index (SECI) identifies the configuration of the software life cycle environment. This index is written to aid reproduction of the hardware and software life cycle environment, for software regeneration, reverification, or software modification, and should:

a. Identify the software life cycle environment hardware and its operating system software.

b. Identify the software development tools, such as compilers, linkage editors and loaders, and data integrity tools (such as tools that calculate and embed checksums or cyclical redundancy checks).

c. Identify the test environment used to verify the software product, for example, the software verification tools.

d. Identify qualified tools and their associated tool qualification data.

*NOTE: This data may be included in the Software Configuration Index.*

## 11.16 SOFTWARE CONFIGURATION INDEX

The Software Configuration Index (SCI) identifies the configuration of the software product.

*NOTE: The SCI can contain one data item or a set (hierarchy) of data items. The SCI can contain the items listed below or it may reference another SCI or other configuration identified data that specifies the individual items and their versions.*

## 11.13 JEUX ET PROCEDURES DE VERIFICATION DU LOGICIEL

Les Jeux et Procédures de Vérification du Logiciel détaillent la manière dont les activités du processus de vérification du logiciel sont réalisées. Ces données doivent décrire :

a. Procédures de revue et d'analyse : Détails complémentaires de la description dans le Plan de Vérification du Logiciel décrivant l'étendue et la profondeur des méthodes de revue ou d'analyse à utiliser.

b. Jeux de test : But de chaque jeu de test, ensemble des données d'entrée, conditions et résultats attendus pour l'obtention des critères de couverture exigés ainsi que les critères de réussite ou d'échec.

c. Procédures de test : Instructions pas à pas sur la manière de préparer et d'exécuter chaque test, sur la manière d'évaluer les résultats et sur l'environnement de test à utiliser.

## 11.14 RESULTATS DE VERIFICATION DU LOGICIEL

Les Résultats de Vérification du Logiciel sont générés par les activités du processus de vérification du logiciel. Les Résultats de Vérification du Logiciel doivent :

a. Pour chaque revue, analyse, et test, indiquer les procédures qui ont satisfait les critères de réussite et celles qui ne l'ont pas, et donner les résultats définitifs de réussite et d'échec.

b. Identifier l'élément de configuration ou la version de logiciel soumis à revue, analyse, ou test.

c. Comprendre les résultats des tests, revues, et analyses, y compris les analyses de couverture et de traçabilité.

## 11.15 REPERTOIRE DE LA CONFIGURATION DE L'ENVIRONNEMENT DU LOGICIEL

Le Répertoire de la Configuration de l'Environnement du Logiciel (SECI) identifie la configuration de cet environnement. Ce répertoire est écrit pour aider à reconstituer l'environnement matériel et logiciel, dans le but de régénérer, revérifier, ou modifier le logiciel, et doit :

a. Identifier l'environnement matériel du cycle de vie du logiciel et son système d'exploitation.

b. Identifier les outils de développement du logiciel, tels que compilateurs, éditeurs de liens et chargeurs, et outils relatifs à l'intégrité des données (tels que les outils qui calculent et intègrent des sommes de contrôle ou des contrôles de redondance cycliques).

c. Identifier l'environnement de test utilisé pour la vérification du produit logiciel, par exemple les outils de vérification du logiciel.

d. Identifier les outils qualifiés et leurs données de qualification associées.

*NOTA : On peut inclure ces données dans le Répertoire de la Configuration du Logiciel.*

## 11.16 REPERTOIRE DE LA CONFIGURATION DU LOGICIEL

Le Répertoire de la Configuration du Logiciel (SCI) identifie la configuration du produit logiciel.

*NOTA : Le SCI peut être constitué d'une seule unité ou être formé d'un ensemble (hiérarchie) d'unités d'information. Le SCI peut comprendre tous les éléments énumérés ci-dessous ou faire référence à un autre SCI ou à d'autres unités gérées en configuration qui spécifient les éléments et leur* version.

The SCI should identify:

a. The software product.

b. Executable Object Code.

c. Each Source Code component.

d. Previously developed software in the software product, if used.

e. Software life cycle data.

f. Archive and release media.

g. Instructions for building the Executable Object Code, including, for example, instructions and data for compiling and linking; and the procedures used to recover the software for regeneration, testing or

h. Reference to the Software Life Cycle Environment Configuration Index (subsection 11.15), if it is packaged separately.

i. Data integrity checks for the Executable Object Code, if used.

*NOTE: The SCI may be produced for one software product version or it may be extended to contain data for several alternative or successive software product versions.*

## 11.17 PROBLEM REPORTS

Problem reports are a means to identify and record the resolution to software product anomalous behavior, process non-compliance with software plans and standards, and deficiencies in software life cycle data. Problem reports should include:

a. Identification of the configuration item and/or the software life cycle process activity in which the problem was observed

b. Identification of the configuration item(s) to be modified or a description of the process to be changed.

c. A problem description that enables the problem to be understood and resolved.

d. A description of the corrective action taken to resolve the reported problem.

## 11.18 SOFTWARE CONFIGURATION MANAGEMENT RECORDS

The results of the SCM process activities are recorded in SCM Records. Examples include configuration identification lists, baseline or software library records, change history reports, archive records, and release records. These examples do not imply records of these specific types need to be produced.

*NOTE: Due to the integral nature of the SCM process, its outputs will often be included as parts of other software life cycle data.*

## 11.19 SOFTWARE QUALITY ASSURANCE RECORDS

The results of the SQA process activities are recorded in SQA Records. These may include SQA review or audit reports, meeting minutes, records of authorized process deviations, or software conformity review records.

Le SCI doit identifier :

a. Le produit logiciel.

b. Le Code Objet Exécutable.

c. Chaque composant du Code Source.

d. Les logiciels développés antérieurement inclus dans le produit logiciel, si on en utilise.

e. Les données du cycle de vie du logiciel.

f. Les supports d'archivage et de mise à disposition.

g. Les instructions pour générer le Code Objet Exécutable, y compris, par exemple, les instructions et les données de compilation et d'édition de liens, et les procédures utilisées pour récupérer le logiciel à des fins de régénération, test, ou modification.

h. La référence au Répertoire de la Configuration de l'Environnement du Logiciel (sous-section 11.15), s'il est séparé.

i. Les contrôles d'intégrité de données pour le Code Objet Exécutable, en cas d'utilisation.

*NOTA :* *Le SCI peut être généré pour une version du logiciel, ou peut être étendu de façon à contenir* les données de plusieurs versions différentes ou successives du logiciel.

## 11.17 RAPPORTS D'ANOMALIES

Les rapports d'anomalies constituent un moyen d'identifier les dysfonctionnements du logiciel, les non-conformités de processus vis à vis des plans et règles, les insuffisances des données du cycle de vie et d'en enregistrer la solution. Les rapports d'anomalies doivent comporter :

a. L'identification de l'élément de configuration et/ou de l'activité du cycle de vie du logiciel dans lequel l'anomalie a été observée.

b. L'identification de l' (ou des) élément(s) de configuration à modifier ou une description du processus à modifier.

c. Une description de l'anomalie qui permette de la comprendre et de la résoudre.

d. Une description de l'action corrective entreprise pour résoudre l'anomalie constatée.

## 11.18 DOCUMENTS DE GESTION DE CONFIGURATION DU LOGICIEL

Les produits des activités du processus de SCM sont enregistrés dans les Documents SCM. Ceux-ci comprennent par exemple des listes d'identification de configuration, des registres relatifs aux référentiels ou à la bibliothèque du logiciel, des comptes rendus d'historique des modifications, des registres d'archivage et des registres de mise à disposition. Ces exemples n'impliquent pas nécessairement que de tels documents soient établis.

*NOTA :* *En raison de la nature intégrale du processus de SCM, ses sorties seront souvent incluses dans d'autres données* du cycle de vie du logiciel.

## 11.19 DOCUMENTS D'ASSURANCE QUALITE DU LOGICIEL

Les produits des activités du processus de SQA sont enregistrés dans les Documents SQA. Ceux-ci comprennent les comptes rendus des revues ou audits de SQA, les procès-verbaux de réunions, des relevés de dérogations accordées pour des processus, ou les procès-verbaux de revue de conformité du logiciel.

## 11.20 SOFTWARE ACCOMPLISHMENT SUMMARY

The Software Accomplishment Summary is the primary data item for showing compliance with the Plan for Software Aspects of Certification. This summary should include:

a. System overview: This section provides an overview of the system, including a description of its functions and their allocation to hardware and software, the architecture, the processor(s) used, the hardware/software interfaces, and safety features. This section also describes any differences from the system overview in the Plan for Software Aspects of

b. Software overview: This section briefly describes the software functions with emphasis on the safety and partitioning concepts used, and explains differences from the software overview proposed in the Plan for Software Aspects of Certification.

c. Certification considerations: This section restates the certification considerations described in the Plan for Software Aspects of Certification and describes any differences.

d. Software characteristics: This section states the Executable Object Code size, timing and memory margins, resource limitations, and the means of measuring each characteristic.

e. Software life cycle: This section summarizes the actual software life cycle(s) and explains differences from the software life cycle and software life cycle processes proposed in the Plan for Software Aspects of Certification.

f. Software life cycle data: This section references the software life cycle data produced by the software development processes and integral processes. It describes the relationship of the data to each other and to other data defining the system, and the means by which software life cycle data will be made available to the certification authority. This section also describes any differences from the Plan for Software Aspects of Certification.

g. Additional considerations: This section summarizes certification issues that may warrant the attention of the certification authority and references data items applicable to these issues, such as issue papers or special conditions.

h. Software identification: This section identifies the software configuration by part number and version.

i. Change history: If applicable, this section includes a summary of software changes with attention to changes made due to failures affecting safety, and identifies changes from the software life cycle processes since the previous certification.

j. Software status: This section contains a summary of problem reports unresolved at the time of certification, including a statement of functional limitations.

k. Compliance statement: This section includes a statement of compliance with this document and a summary of the methods used to demonstrate compliance with criteria specified in the software plans. This section also addresses additional rulings and deviations from the software plans, standards and this document.

## 11.20 RESUME DES TRAVAUX REALISES

Le Résumé des Travaux Réalisés constitue la principale unité d'information pour la démonstration de la conformité avec le Plan des Aspects Logiciels de la Certification. Ce résumé doit comporter les éléments suivants :

a. Vue d'ensemble du système : Cette section fournit une vue d'ensemble du système, y compris une description de ses fonctions et de leur allocation au matériel et au logiciel, de l'architecture, du(des) processeur(s) utilisé(s), des interfaces matériel/logiciel, et des caractéristiques liées à la sécurité. Cette section décrit aussi toutes les différences avec la vue d'ensemble du système figurant dans le Plan des Aspects Logiciels de la Certification.

b. Vue d'ensemble du logiciel : Cette section décrit brièvement les fonctions du logiciel en insistant sur les concepts employés pour la sécurité et le partitionnement et explique les différences avec la vue d'ensemble du logiciel proposée dans le Plan des Aspects Logiciels de la Certification.

c. Considérations relatives à la certification : Cette section reprend les considérations relatives à la certification, décrites dans le Plan des Aspects Logiciels de la Certification, et décrit toutes les différences.

d. Caractéristiques du logiciel : Cette section précise la taille du Code Objet Exécutable, les marges de temps d'exécution et d'occupation mémoire, les limitations des ressources, et les moyens de mesure de chaque caractéristique.

e. Cycle de vie du logiciel : Cette section résume le(s) cycle(s) de vie du logiciel effectif(s) et explique les différences avec le cycle de vie et ses processus, proposés dans le Plan des Aspects Logiciels de la Certification.

f. Données du cycle de vie du logiciel : Cette section référence les données du cycle de vie générées par le processus de développement du logiciel et les processus intégraux. Elle décrit les liens entre données ou avec celles définissant le système, et les moyens par lesquels ces données du cycle de vie du logiciel sont mises à la disposition de l'Autorité de certification. Cette section décrit aussi toutes les différences avec le Plan des Aspects Logiciels de la Certification.

g. Considérations complémentaires : Cette section résume les questions de certification qui peuvent justifier l'attention particulière de l'Autorité de certification et référence les documents ayant trait à ces questions, tels que conditions spéciales de certification ("Certification Review Items / Issue Papers").

h. Identification du logiciel : Cette section identifie la configuration du logiciel par numéro de référence et version.

i. Historique des modifications : Si nécessaire, cette section comprend un résumé des modifications du logiciel en insistant sur les modifications consécutives à des pannes touchant à la sécurité, et identifie les changements touchant les processus du cycle de vie depuis la dernière certification.

j. Etat du logiciel : Cette section contient un résumé des rapports d'anomalies non résolus au moment de la certification, y compris une déclaration des limitations fonctionnelles.

k. Déclaration de conformité : Cette section comprend une déclaration de conformité avec ce document et un résumé des méthodes utilisées pour démontrer la conformité avec les critères spécifiés dans les plans. Cette section traite aussi des règles additionnelles et des écarts vis à vis des plans et règles du logiciel et de ce document.

# SECTION 12

# ADDITIONAL CONSIDERATIONS

The previous sections of this document provide guidance for satisfying certification requirements in which the applicant submits evidence of the software life cycle processes. This section introduces additional considerations concerning the software aspects of certification in relation to the use of previously developed software, qualification of software tools, and alternative methods for achieving the objectives of the previous sections of this document.

## 12.1 USE OF PREVIOUSLY DEVELOPED SOFTWARE

The guidelines of this subsection discuss the issues associated with the use of previously developed software, including the assessment of modifications, the effect of changing an aircraft installation, application environment, or development environment, upgrading a development baseline, and SCM and SQA considerations. The intention to use previously developed software is stated in the Plan for Software Aspects of Certification.

### 12.1.1 Modifications to Previously Developed Software

This guidance discusses modifications to previously developed software where the outputs of the previous software life cycle processes comply with this document. Modification may result from requirement changes, the detection of errors, and/or software enhancements. Analysis activities for proposed modifications include:

a. The revised outputs of the system safety assessment process should be reviewed considering the proposed modifications.

b. If the software level is revised, the guidelines of paragraph 12.1.4 should be considered.

c. Both the impact of the software requirements changes and the impact of software architecture changes should be analyzed, including the consequences of software requirement changes upon other requirements and coupling between several software components that may result in reverification effort involving more than the modified area.

d. The area affected by a change should be determined. This may be done by data flow analysis, control flow analysis, timing analysis and traceability analysis.

e. Areas affected by the change should be reverified considering the guidelines of section 6.

### 12.1.2 Change of Aircraft Installation

Airborne systems or equipment containing software which has been previously certified at a certain software level and under a specific certification basis may be used in a new aircraft installation. This guidance should be used for new aircraft installations, if using previously developed software:

a. The system safety assessment process assesses the new aircraft installation and determines the software level and the certification basis. No additional effort will be required if these are the same for the new installation as they were in the previous installation.

b. If functional modifications are required for the new installation, the guidelines of paragraph 12.1.1, Modifications to Previously Developed Software, should be satisfied.

# SECTION 12

# CONSIDERATIONS COMPLEMENTAIRES

Les sections précédentes de ce document donnent des recommandations pour satisfaire les exigences de certification pour lesquelles le postulant soumet les preuves issues du cycle de vie du logiciel. Cette section présente des considérations complémentaires relatives aux aspects logiciels de la certification pour l'utilisation de logiciel développé antérieurement, la qualification des outils, et l'utilisation de méthodes de substitution pour satisfaire les objectifs présentés dans les sections précédentes de ce document.

## 12.1 UTILISATION DE LOGICIEL DEVELOPPE ANTERIEUREMENT

Les recommandations de cette sous-section traitent des questions liées à l'utilisation de logiciel développé antérieurement, comprenant notamment l'évaluation des modifications, l'effet de changement de l'installation d'un aéronef, de l'environnement d'utilisation ou de développement, de l'évolution d'un référentiel de développement, ainsi que les aspects gestion de configuration et assurance qualité correspondants. L'intention d'utiliser du logiciel développé antérieurement est déclarée dans le Plan des Aspects Logiciels de la Certification.

### 12.1.1 Modifications d'un logiciel développé antérieurement

Ces directives traitent des modifications apportées à du logiciel développé antérieurement, dont le cycle de vie a fourni des produits conformes à ce document. Les modifications peuvent résulter de modifications de spécifications, de détection d'erreurs, et/ou d'améliorations du logiciel. L'analyse des modifications proposées comprend les points suivants :

a. Les résultats révisés de l'analyse de sécurité du système doivent être soumis à revue en prenant en considération les modifications proposées.

b. Si le niveau logiciel est modifié, les recommandations du paragraphe 12.1.4. doivent être prises en considération.

c. On doit analyser à la fois l'impact des modifications des spécifications et de l'architecture du logiciel, y compris notamment les conséquences des modifications des spécifications du logiciel sur d'autres exigences et du couplage entre plusieurs composants logiciels qui peuvent entraîner un travail de revérification impliquant plus que la partie modifiée.

d. On doit déterminer la partie affectée par une modification. Ceci peut se faire par analyse des flux de données, analyse des flux de contrôle, analyse de l'ordonnancement et des temps d'exécution, et analyse de traçabilité.

e. On doit revérifier les parties affectées par la modification en prenant en considération les recommandations de la section 6.

### 12.1.2 Modification de l'installation d'un aéronef

Des systèmes ou équipements de bord homologués antérieurement avec un certain niveau logiciel et selon des bases spécifiques de certification peuvent être utilisés dans une nouvelle installation. Dans ce cas, les directives suivantes doivent être prises en compte pour de nouvelles installations :

a. L'analyse de sécurité du système évalue la nouvelle installation et détermine le niveau logiciel et la base de certification. S'ils sont identiques pour la nouvelle et l'ancienne installation aucun effort supplémentaire ne sera nécessaire.

b. Si des modifications fonctionnelles sont nécessaires pour la nouvelle installation, on doit satisfaire les recommandations du paragraphe 12.1.1, Modifications de logiciel développé antérieurement.

c. If the previous development activity did not produce outputs required to substantiate the safety objectives of the new installation, the guidelines of paragraph 12.1.4, Upgrading A Development Baseline, should be satisfied.

12.1.3 Change of Application or Development Environment

Use and modification of previously developed software may involve a new development environment, a new target processor or other hardware, or integration with other software than that used for the original application.

New development environments may increase or reduce some activities within the software life cycle. New application environments may require activities in addition to software life cycle process activities which address modifications. Guidance for change of application or development environment includes:

a. If a new development environment uses software development tools, the guidelines of subsection 12.2, Tool Qualification, may be

b. The rigor of the evaluation of an application change should consider the complexity and sophistication of the programming language. For example, the rigor of the evaluation for Ada generics will be greater if the generic parameters are different in the new application. For object oriented languages, the rigor will be greater if the objects that are inherited are different in the new application

c. If a different compiler or different set of compiler options are used, resulting in different object code, the results from a previous software verification process activity using the object code may not be valid and should not be used for the new application. In this case, previous test results may no longer be valid for the structural coverage criteria of the new application. Similarly, compiler assumptions about optimization may not be valid.

d. If a different processor is used then:

   (1) The results from a previous software verification process activity directed at the hardware/software interface should not be used for the new application.

   (2) The previous hardware/software integration tests should be executed for the new application.

   (3) Reviews of hardware/software compatibility should be repeated.

   (4) Additional hardware/software integration tests and reviews may be necessary.

a. Verification of software interfaces should be conducted where previously developed software is used with different interfacing software.

12.1.4 Upgrading A Development Baseline

Guidelines follow for software whose software life cycle data from a previous application are determined to be inadequate or do not satisfy the objectives of this document, due to the safety objectives associated with a new application. These guidelines are intended to aid in the acceptance of:

- Commercial off-the-shelf software.
- Airborne software developed to other guidelines.
- Airborne software developed prior to the existence of this document.
- Software previously developed to this document at a lower software level.

c. Si l'activité antérieure de développement n'a pas généré les produits nécessaires pour satisfaire les objectifs de sécurité de la nouvelle installation, les recommandations du paragraphe 12.1.4, Amélioration d'un référentiel s'appliquent.

### 12.1.3 Modification de l'environnement de développement ou de son utilisation

L'utilisation et la modification de logiciel développé antérieurement peut impliquer un nouvel environnement de développement, une nouvelle machine cible ou un nouveau matériel, ou l'intégration avec un autre logiciel que celui employé dans l'application d'origine.

De nouveaux environnements de développement peuvent augmenter ou réduire certaines activités du cycle de vie du logiciel. De nouveaux environnements d'utilisation peuvent exiger des activités en supplément de celles du traitement des modifications prévues dans le cycle de vie du logiciel. Les directives pour les modifications d'environnement d'utilisation ou de développement sont les suivantes :

a. Si un nouvel environnement de développement comprend des outils de développement de logiciel, les recommandations de la sous-section 12.2 Qualification d'outil peuvent être applicables.

b. La rigueur de l'évaluation d'une modification de l'utilisation doit prendre en considération la complexité et la sophistication du langage de programmation. Par exemple, pour des unités génériques ADA, la rigueur de l'évaluation sera plus importante si les paramètres génériques sont différents dans la nouvelle utilisation. Pour les langages orientés objet, la rigueur sera plus importante si les objets hérités sont différents dans la nouvelle utilisation.

c. Dans le cas d'utilisation d'un compilateur différent ou d'options de compilation différentes, avec pour conséquence un code objet différent, les résultats d'une activité antérieure de vérification nécessitant le code objet peuvent ne pas être valables et ne doivent pas être utilisés pour la nouvelle application. Dans ce cas, les résultats de tests antérieurs peuvent ne pas remplir les critères de couverture structurelle dans le cadre de la nouvelle utilisation. De même, les hypothèses concernant l'optimisation du compilateur peuvent ne pas être valables.

d. Dans le cas d'utilisation d'un processeur différent :

   (1) Les résultats d'une activité antérieure de vérification concernant l'interface matériel/logiciel ne doivent pas être utilisés pour la nouvelle application.

   (2) Les tests d'intégration matériel/logiciel antérieurs doivent être exécutés pour la nouvelle application.

   (3) Les revues de la compatibilité matériel/logiciel doivent être reconduites.

   (4) Des revues et des tests d'intégration matériel/logiciel complémentaires pourront être nécessaires.

e. Une vérification des interfaces du logiciel doit être effectuée si le logiciel développé antérieurement est intégré avec un logiciel différent.

### 12.1.4 Mise à niveau d'un référentiel

Les recommandations suivantes concernent les logiciels dont on a déterminé que les données du cycle de vie, issues d'une utilisation antérieure, sont inadéquates ou ne satisfont pas les objectifs de ce document, en raison des objectifs de sécurité liés à une nouvelle utilisation. Ces recommandations ont pour but d'aider à l'acceptation :

- De logiciels du commerce sur étagère.
- De logiciels embarqués développés selon d'autres recommandations.
- De logiciels embarqués développés antérieurement à l'existence de ce document.
- De logiciels développés selon ce document, mais à un niveau logiciel inférieur.

Guidance for upgrading a development baseline includes:

a. The objectives of this document should be satisfied while taking advantage of software life cycle data of the previous development that satisfy the objectives for the new application

b. Software aspects of certification should be based on the failure conditions and software level(s) as determined by the system safety assessment process. Comparison to failure conditions of the previous application will determine areas which may need to be upgraded.

c. Software life cycle data from a previous development should be evaluated to ensure that the software verification process objectives of the software level are satisfied for the new application.

d. Reverse engineering may be used to regenerate software life cycle data that is inadequate or missing in satisfying the objectives of this document. In addition to producing the software product, additional activities may need to be performed to satisfy the software verification process objectives.

e. If use of product service history is planned to satisfy the objectives of this document in upgrading a development baseline, the guidelines of paragraph 12.3.5 should be considered.

f. The applicant should specify the strategy for accomplishing compliance with this document in the Plan for Software Aspects of Certification.

### 12.1.5 Software Configuration Management Considerations

If previously developed software is used, the software configuration management process for the new application should include, in addition to the guidelines of section 7:

a. Traceability from the software product and software life cycle data of the previous application to the new application.

b. Change control that enables problem reporting, problem resolution, and tracking of changes to software components used in more than one application.

### 12.1.6 Software Quality Assurance Considerations

If previously developed software is used, the software quality assurance process for the new application should include, in addition to the guidelines of section 8:

a. Assurance that the software components satisfy or exceed the software life cycle criteria of the software level for the new application.

b. Assurance that changes to the software life cycle processes are stated on the software plans.

## 12.2 TOOL QUALIFICATION

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

Les directives pour la mise à niveau d'un référentiel sont les suivantes :

a. Les objectifs de ce document doivent être satisfaits en profitant des données du cycle de vie du développement antérieur qui satisfont les objectifs pour la nouvelle utilisation.

b. Les aspects logiciels de la certification doivent être fondés sur les conditions de panne et le(s) niveau(x) logiciel(s) selon la détermination faite par l'analyse de sécurité du système. La comparaison avec les conditions de panne relatives à l'utilisation précédente déterminera les domaines qu'il peut être nécessaire de mettre à niveau.

c. Les données du cycle de vie d'un développement antérieur doivent être évaluées afin de s'assurer que les objectifs du processus de vérification pour le niveau logiciel associé à la nouvelle utilisation sont satisfaits.

d. Des techniques de rétroingéniérie peuvent être utilisées pour régénérer les données de cycle de vie de logiciel inadéquates ou manquantes pour satisfaire les objectifs de ce document. Outre la réalisation du produit logiciel, il pourra être nécessaire d'effectuer d'autres activités afin de satisfaire les objectifs de vérification du logiciel.

e. Si l'utilisation d'un historique du produit en service est prévu dans le but de satisfaire les objectifs de ce document lors de la mise à niveau d'un référentiel, les recommandations du paragraphe 12.3.5. doivent être prises en compte.

f. Le postulant doit spécifier la stratégie employée pour obtenir la conformité avec ce document dans le Plan des Aspects Logiciels de la Certification.

### 12.1.5 Considérations relatives à la gestion de configuration

Si on utilise un logiciel développé antérieurement, en plus des recommandations de la section 7, la gestion de configuration doit assurer :

a. La traçabilité du produit logiciel et des données du cycle de vie entre l'application précédente et la nouvelle application.

b. Un contrôle des modifications qui permette le compte rendu des anomalies, la résolution des problèmes ainsi que le suivi des modifications sur des composants logiciels utilisés dans plus d'une application.

### 12.1.6 Considérations relatives à l'assurance qualité

Dans le cas d'utilisation de logiciel développé antérieurement, le processus d'assurance qualité du logiciel pour la nouvelle utilisation doit comprendre, outre les recommandations de la section 8 :

a. L'assurance que les composants logiciels satisfont au minimum les critères correspondant au niveau logiciel associé à la nouvelle utilisation.

b. L'assurance que les modifications apportées aux processus du cycle de vie figurent dans les plans.

## 12.2 QUALIFICATION D'OUTIL

La qualification d'outil est nécessaire quand des processus de ce document sont éliminés, réduits, ou automatisés par l'utilisation d'un outil logiciel dont les sorties ne sont pas vérifiées selon les recommandations de la section 6. L'utilisation d'outils logiciels pour automatiser les activités des processus du cycle de vie du logiciel peut aider à satisfaire les objectifs de sécurité du système dans la mesure où ils peuvent renforcer la conformité avec les règles de développement du logiciel et effectuer automatiquement les vérifications.

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. If partitioning of tool functions can be demonstrated, only those functions that are used to eliminate, reduce, or automate software life cycle process activities, and whose outputs are not verified, need be qualified.

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

Software tools can be classified as one of two types:

- Software development tools: Tools whose output is part of airborne software and thus can introduce errors. For example, a tool which generates Source Code directly from low-level requirements would have to be qualified if the generated Source Code is not verified as specified in section 6.
- Software verification tools: Tools that cannot introduce errors, but may fail to detect them. For example, a static analyzer, that automates a software verification process activity, should be qualified if the function that it performs is not verified by another activity. Type checkers, analysis tools and test tools are other examples.

Tool qualification guidance includes:

a. Tools should be qualified according to the type specified above.

b. Combined software development tools and software verification tools should be qualified to comply with the guidelines in paragraph 12.2.1, unless partitioning between the two functions can be demonstrated.

c. The software configuration management process and software quality assurance process objectives for airborne software should apply to software tools to be qualified.

The software verification process objectives for software development tools are described in paragraph 12.2.1, item d.

A tool may be qualified only for use on a specific system where the intention to use the tool is stated in the Plan for Software Aspects of Certification. Use of the tool for other systems may need further qualification.

### 12.2.1 Qualification Criteria for Software Development Tools

The qualification criteria for software development tools includes:

a. If a software development tool is to be qualified, the software development processes for the tool should satisfy the same objectives as the software development processes of airborne software.

b. The software level assigned to the tool should be the same as that for the airborne software it produces, unless the applicant can justify a reduction in software level of the tool to the certification authority.

*NOTE: A reduction in a tool's software level can be based upon the significance of the software verification process activity to be eliminated, reduced or automated, with respect to the entire suite of verification activities. This significance is a function of:*

- *The type of software verification process activity to be eliminated, reduced or automated. For example, a verification activity for conformance of the Source Code with software indentation standards is less significant than verification activity for compliance of the Executable Object Code with the high-level requirements.*

Le processus de qualification d'outil a pour but de garantir que l'outil donne une confiance au moins équivalente à celle du (ou des) processus éliminé(s), réduits, ou automatisés. Si on peut démontrer le partitionnement des fonctions de l'outil, il ne sera nécessaire de qualifier que les fonctions utilisées pour éliminer, réduire, ou automatiser les activités des processus du cycle de vie du logiciel, et dont les sorties ne sont pas vérifiées.

On ne peut qualifier que des outils déterministes, c'est-à-dire des outils qui, fonctionnant dans le même environnement, génèrent la même sortie pour les mêmes données d'entrée. Le processus de qualification d'outil peut s'appliquer soit à un outil unique, soit à un ensemble d'outils.

Les outils logiciels peuvent être classés selon deux types :

- Outils de développement du logiciel : Outils dont la sortie est intégrée au logiciel embarqué et qui peuvent de ce fait introduire des erreurs. Par exemple, un outil qui génère directement un Code Source à partir d'exigences de bas niveau doit être qualifié si le Code Source généré n'est pas vérifié selon les recommandations de la section 6.
- Outils de vérification du logiciel : Outils qui ne peuvent pas introduire d'erreurs, mais qui peuvent être incapables de les détecter. Par exemple un analyseur statique, qui automatise une activité d'un processus de vérification du logiciel, doit être qualifié si la fonction qu'il exécute n'est pas vérifiée par une autre activité. Des contrôleurs du type des données, des outils d'analyse, et des outils de test constituent d'autres exemples.

Les directives en matière de qualification d'outil comprennent en particulier :

a. Les outils doivent être qualifiés en fonction du type spécifié ci-dessus.

b. Les combinaisons d'outils de développement et de vérification doivent être qualifiées conformément aux recommandations du paragraphe 12.2.1, sauf si la démonstration du partitionnement entre les deux est possible.

c. Les mêmes objectifs de gestion de configuration et d'assurance qualité applicables aux logiciels embarqués doivent s'appliquer aux outils à qualifier.

Les objectifs de vérification pour les outils de développement sont décrits dans le paragraphe 12.2.1 d.

On ne peut qualifier un outil que pour une utilisation sur un système spécifique et déclarée dans le Plan des Aspects Logiciels de la Certification. L'utilisation de cet outil pour d'autres systèmes peut nécessiter une qualification supplémentaire.

### 12.2.1 Critères de qualification des outils de développement

Les critères de qualification des outils de développement sont :

a. Lorsqu'un outil de développement doit être qualifié, il doit satisfaire les mêmes objectifs que ceux applicables au logiciel embarqué.

b. Le niveau logiciel affecté à l'outil doit être le même que celui du logiciel embarqué qu'il produit, sauf si le postulant peut justifier à l'Autorité de certification une réduction du niveau logiciel de l'outil.

*NOTA : Une réduction du niveau logiciel pour un outil peut être basée sur l'importance de l'activité de vérification à éliminer, réduire, ou automatiser, par rapport à la suite complète des activités de vérification. Cette importance est fonction :*

- *du type d'activité de vérification à éliminer, réduire ou automatiser. Par exemple, une activité de vérification de la conformité du Code Source avec les règles d'indentation du logiciel est moins significative qu'une activité de vérification de la conformité du Code Objet Exécutable avec les exigences de haut niveau.*

- *The likelihood that other verification activities would have detected the same error(s).*

c. The applicant should demonstrate that the tool complies with its Tool Operational Requirements (subparagraph 12.2.3.2). This demonstration may involve a trial period during which a verification of the tool output is performed and tool-related problems are analyzed, recorded and corrected.

d. Software development tools should be verified to check the correctness, consistency, and completeness of the Tool Operational Requirements and to verify the tool against those requirements. The objectives of the tool's software verification process are different from those of the airborne software since the tool's high-level requirements correspond to its Tool Operational Requirements instead of system requirements. Verification of software development tools may be achieved by:

(1) Review of the Tool Operational Requirements as described in paragraph 6.3.1, items a and b.

(2) Demonstration that the tool complies with its Tool Operational Requirements under normal operating conditions.

(3) Demonstration that the tool complies with its Tool Operational Requirements while executing in abnormal operating conditions, including external disturbances and selected failures applied to the tool and its environment.

(4) Requirements-based coverage analysis and additional tests to complete the coverage of the requirements.

(5) Structural coverage analysis appropriate for the tool's software level.

(6) Robustness testing for tools with a complex data flow or control flow, as specified in subparagraph 6.4.2.2, appropriate to the tool's software level.

(7) Analysis of potential errors produced by the tool, to confirm the validity of the Tool Qualification Plan.

12.2.2 Qualification Criteria for Software Verification Tools

The qualification criteria for software verification tools should be achieved by demonstration that the tool complies with its Tool Operational Requirements under normal operational conditions.

12.2.3 Tool Qualification Data

Guidance for tool qualification data includes:

a. When qualifying a tool, the Plan for Software Aspects of Certification of the related airborne software should specify the tool to be qualified and reference the tool qualification data

b. The tool qualification data should be controlled as Control Category 1 (CC1) for software development tools and CC2 for software verification tools.

c. For software development tools, the tool qualification data should be consistent with the data in section 11 and have the same characteristics and content as data for airborne software, with these considerations:

(1) A Tool Qualification Plan satisfies the same objectives as the Plan for Software Aspects of Certification of the airborne software.

(2) Tool Operational Requirements satisfies the same objectives as the Software Requirements Data of the airborne software.

- *du degré de probabilité que d'autres activités de vérification détectent la (ou les) même(s) erreur(s).*

c. Le postulant doit démontrer que l'outil est conforme aux Spécifications Opérationnelles de l'Outil (sous-paragraphe 12.2.3.2). Cette démonstration peut impliquer une période probatoire pendant laquelle les sorties de l'outil sont vérifiées et les problèmes attribuables à l'outil sont analysés, enregistrés et corrigés.

d. Les outils de développement doivent être vérifiés de manière à contrôler l'exactitude, la cohérence, et l'exhaustivité des Spécifications Opérationnelles de l'Outil et à vérifier l'outil par rapport à ces spécifications. Les objectifs du processus de vérification du logiciel de l'outil diffèrent de ceux du logiciel embarqué du fait que les exigences de haut niveau de l'outil correspondent aux Spécifications Opérationnelles de l'Outil plutôt qu'aux spécifications du système. La vérification des outils de développement du logiciel est fondée sur :

(1) La revue des Spécifications Opérationnelles de l'Outil selon la description du paragraphe 6.3.1 a et b.

(2) La démonstration de la conformité de l'outil avec ses Spécifications Opérationnelles dans les conditions normales de fonctionnement.

(3) La démonstration de la conformité de l'outil avec ses Spécifications Opérationnelles dans des conditions anormales de fonctionnement, comprenant notamment des perturbations extérieures et des pannes choisies appliquées à l'outil et à son environnement.

(4) L'analyse de la couverture des spécifications et les tests complémentaires ayant pour but de compléter cette couverture.

(5) L'analyse de la couverture structurelle adaptée au niveau logiciel de l'outil.

(6) Pour des outils dont les flux de données ou de contrôle sont complexes, les tests de robustesse réalisés selon les recommandations du sous-paragraphe 6.4.2.2, et adaptés au niveau logiciel de l'outil.

(7) L'analyse des erreurs susceptibles d'être générées par l'outil, afin de confirmer la validité du Plan de Qualification de l'Outil.

12.2.2 Critères de qualification des outils de vérification

Les critères de qualification des outils de vérification sont satisfaits en démontrant que l'outil est conforme à ses Spécifications Opérationnelles dans les conditions normales de fonctionnement.

12.2.3 Données de qualification d'outil

Les directives concernant les données de qualification d'outil comprennent :

a. Pour la qualification d'un outil, le Plan des Aspects Logiciels de la Certification du logiciel embarqué concerné doit préciser l'outil à qualifier et référencer les données de qualification de cet outil.

b. La catégorie de contrôle applicable aux données de qualification d'un outil est la Catégorie de Contrôle 1 (CC1) pour les outils de développement et la Catégorie de Contrôle 2 (CC2) pour les outils de vérification.

c. Pour les outils de développement, les données de qualification doivent être cohérentes avec celles de la section 11 et avoir les mêmes caractéristiques et le même contenu que les données relatives au logiciel embarqué, sachant que :

(1) Un Plan de Qualification de l'Outil satisfait les mêmes objectifs que le Plan des Aspects Logiciels de la Certification du logiciel embarqué.

(2) Les Spécifications Opérationnelles de l'Outil satisfont les mêmes objectifs que les Spécifications du Logiciel pour le logiciel embarqué.

(3) A Tool Accomplishment Summary satisfies the same objectives as the Software Accomplishment Summary of the airborne software.

12.2.3.1 Tool Qualification Plan

For software development tools to be qualified, the Tool Qualification Plan describes the tool qualification process. This plan should include:

a. Configuration identification of the tool

b. Details of the certification credit sought, that is, the software verification process activities to be eliminated, reduced or automated.

c. The software level proposed for the tool.

d. A description of the tool's architecture.

e. The tool qualification activities to be performed.

f. The tool qualification data to be produced.

12.2.3.2 Tool Operational Requirements

Tool Operational Requirements describe the tool's operational functionality. This data should include:

a. A description of the tool's functions and technical features. For software development tools, it includes the software development process activities performed by the tool.

b. User information, such as installation guides and user manuals.

c. A description of the tool's operational environment.

d. For software development tools, the expected responses of the tool under abnormal operating conditions.

12.2.4 Tool Qualification Agreement

The certification authority gives its agreement to the use of a tool in two steps:

- For software development tools, agreement with the Tool Qualification Plan. For software verification tools, agreement with the Plan for Software Aspects of Certification of the airborne software.

- For software development tools, agreement with the Tool Accomplishment Summary. For software verification tools, agreement with the Software Accomplishment Summary of the airborne software.

12.3 ALTERNATIVE METHODS

Some methods were not discussed in the previous sections of this document because of inadequate maturity at the time this document was written or limited applicability for airborne software. It is not the intention of this document to restrict the implementation of any current or future methods. Any single alternative method discussed in this subsection is not considered an alternative to the set of methods recommended by this document, but may be used in satisfying one or more of the objectives of in this document.

Alternative methods may be used to support one another. For example, formal methods may assist tool qualification or a qualified tool may assist the use of formal methods.

(3) Un Résumé des Travaux Réalisés pour l'Outil satisfait les mêmes objectifs que le Résumé des Travaux Réalisés du logiciel embarqué.

12.2.3.1 Plan de Qualification de l'Outil

Pour les outils de développement à qualifier, le Plan de Qualification de l'Outil décrit le processus de qualification. Ce plan doit comprendre :

a. L'identification de la configuration de l'outil.

b. Des détails sur le crédit de certification recherché, c'est-à-dire les activités du processus de vérification à éliminer, réduire, ou automatiser.

c. Le niveau logiciel proposé pour l'outil.

d. Une description de l'architecture de l'outil.

e. Les activités de qualification à réaliser.

f. Les données de qualification à produire.

12.2.3.2 Spécifications Opérationnelles de l'Outil

Les Spécifications Opérationnelles de l'Outil décrivent les fonctions opérationnelles de l'outil. Ces données doivent comprendre :

a. Une description des fonctions et des caractéristiques techniques de l'outil. Pour les outils de développement, les activités de développement du logiciel réalisées par l'outil.

b. Des informations pour l'utilisateur, tels que guides d'installation et manuels utilisateur.

c. Une description de l'environnement opérationnel de l'outil.

d. Pour les outils de développement, le comportement attendu de l'outil dans des conditions anormales de fonctionnement.

12.2.4 Agrément de qualification d'outil

L'Autorité de certification donne son agrément pour l'utilisation d'un outil en deux étapes :

- Pour les outils de développement, agrément du Plan de Qualification de l'Outil. Pour les outils de vérification, agrément du Plan des Aspects Logiciels de la Certification du logiciel embarqué.
- Pour les outils de développement agrément, du Résumé des Travaux Réalisés pour l'Outil. Pour les outils de vérification, agrément du Résumé des Travaux Réalisés du logiciel embarqué.

12.3 METHODES DE SUBSTITUTION

Certaines méthodes n'ont pas été traitées dans les sections précédentes de ce document en raison de leur manque de maturité à l'époque où ce document a été rédigé ou de leur applicabilité limitée pour les logiciels embarqués. Ce document n'a pas pour objectif de limiter l'utilisation de telle ou telle méthode actuelle ou future. Aucune des méthodes de substitution décrites dans cette sous-section ne doit être considérée individuellement comme une alternative à l'ensemble des méthodes recommandées par ce document, mais leur utilisation est possible pour satisfaire un ou plusieurs objectifs de ce document.

Des méthodes de substitution peuvent être utilisées pour se soutenir l'une l'autre. Par exemple, les méthodes formelles peuvent aider à la qualification d'un outil ou inversement un outil qualifié peut aider à l'utilisation de méthodes formelles.

An alternative method cannot be considered in isolation from the suite of software development processes. The effort for obtaining certification credit of an alternative method is dependent on the software level and the impact of the alternative method on the software life cycle processes. Guidance for using an alternative method includes:

a. An alternative method should be shown to satisfy the objectives of this document.

b. The applicant should specify in the Plan for Software Aspects of Certification, and obtain agreement from the certification authority for:

   (1) The impact of the proposed method on the software development processes.

   (2) The impact of the proposed method on the software life cycle data.

   (3) The rationale for use of the alternative method which shows that the system safety objectives are satisfied.

a. The rationale should be substantiated by software plans, processes, expected results, and evidence of the use of the method.

12.3.1 Formal Methods

Formal methods involve the use of formal logic, discrete mathematics, and computer-readable languages to improve the specification and verification of software. These methods could produce an implementation whose operational behavior is known with confidence to be within a defined domain. In their most thorough application, formal methods could be equivalent to exhaustive analysis of a system with respect to its requirements. Such analysis could provide:

- Evidence that the system is complete and correct with respect to its
- Determination of which code, software requirements or software architecture satisfy the next higher level of software requirements.

The goal of applying formal methods is to prevent and eliminate requirements, design and code errors throughout the software development processes. Thus, formal methods are complementary to testing. Testing shows that functional requirements are satisfied and detects errors, and formal methods could be used to increase confidence that anomalous behavior will not occur (for inputs that are out of range) or unlikely to occur.

Formal methods may be applied to software development processes with consideration of these factors:

Levels of the design refinement: The use of formal methods begins by specifying software high-level requirements in a formal specification language and verifying by formal proofs that they satisfy system requirements, especially constraints on acceptable operation. The next lower level of requirements are then shown to satisfy the high-level requirements. Performing this process down to the Source Code provides evidence that the software satisfies system requirements. Application of formal methods can start and stop with consecutive levels of the design refinement, providing evidence that those levels of requirements are specified correctly.

Coverage of software requirements and software architecture: Formal methods may be applied to software requirements that:

- Are safety-related.
- Can be defined by discrete mathematics.
- Involve complex behavior, such as concurrency, distributed processing, redundancy management, and synchronization.

Une méthode de substitution ne doit pas être considérée indépendamment de l'ensemble des processus du cycle de vie du logiciel. L'effort nécessaire à l'obtention d'un crédit de certification pour l'utilisation d'une méthode de substitution dépend du niveau logiciel et de l'impact de cette méthode sur le cycle de vie du logiciel. Les directives d'utilisation d'une méthode de substitution comprennent :

a. La méthode doit satisfaire les objectifs de ce document.

b. Le postulant doit indiquer dans le Plan des Aspects Logiciels de la Certification :

   (1) L'impact de la méthode proposée sur le développement du logiciel.

   (2) L'impact de la méthode proposée sur les données du cycle de vie du logiciel.

   (3) La justification de l'utilisation de cette méthode démontrant que les objectifs de sécurité du système sont satisfaits.

   et obtenir sur ces points l'accord de l'Autorité de certification.

c. Cette justification devra être étayée par des plans, des processus, des résultats attendus, et la preuve de l'utilisation effective de la méthode.

### 12.3.1 Méthodes formelles

Les méthodes formelles impliquent l'utilisation de la logique formelle, de l'algèbre et de langages interprétables par ordinateur afin d'améliorer la spécification et la vérification du logiciel. Ces méthodes peuvent conduire à une mise en oeuvre dont on a l'assurance que le comportement opérationnel reste confiné dans un domaine défini. Dans leur application la plus stricte, les méthodes formelles peuvent être équivalentes à l'analyse exhaustive d'un système en ce qui concerne ses exigences. Une telle analyse peut fournir :

- La preuve que le système est complet et correct vis à vis de ses exigences.
- La détermination de quel code, quelles spécifications du logiciel, ou quelle architecture du logiciel satisfait le niveau immédiatement supérieur d'exigences.

L'utilisation de méthodes formelles a pour but d'éviter et éliminer les erreurs de spécification, de conception, et de codage lors du développement du logiciel. De ce fait, les méthodes formelles sont complémentaires des tests. Les tests montrent que les exigences fonctionnelles sont satisfaites et détectent les erreurs, alors que les méthodes formelles peuvent être utilisées pour augmenter la certitude qu'un dysfonctionnement ne se produira pas (pour des données d'entrée qui sont hors des limites) ou qu'il sera hautement improbable.

Les méthodes formelles peuvent s'appliquer au développement des logiciels en prenant en considération les facteurs suivants :

Niveaux de raffinement de la conception : L'utilisation de méthodes formelles commence par la spécification des exigences de haut niveau du logiciel dans un langage formel et par la vérification au moyen de preuves formelles qu'elles satisfont les spécifications du système, en particulier les contraintes relatives à un fonctionnement acceptable. Il est ensuite démontré que le niveau d'exigences immédiatement inférieur satisfait les exigences de haut niveau. L'exécution de ce processus jusqu'au Code Source fournit la preuve que le logiciel satisfait les spécifications du système. L'utilisation de méthodes formelles peut commencer et se terminer à des niveaux consécutifs de raffinement de la conception, fournissant la preuve que ces niveaux d'exigences sont spécifiés correctement.

Couverture des spécifications et de l'architecture du logiciel : Les méthodes formelles peuvent s'appliquer aux spécifications du logiciel qui :

- Sont liées à la sécurité.
- Peuvent être définies algébriquement.
- Impliquent un fonctionnement complexe, tel que parallélisme, traitement réparti, gestion de redondance, et synchronisation.

These criteria can be used to determine the set of requirements at the level of the design refinement to which the formal methods are applied.

Degree of rigor: Formal methods include these increasingly rigorous levels:

- Formal specification with no proofs.
- Formal specification with manual proofs.
- Formal specification with automatically checked or generated proofs.

The use of formal specifications alone forces requirements to be unambiguous. Manual proof is a well-understood process that can be used when there is little detail. Automatically checked or generated proofs can aid the human proof process and offer a higher degree of dependability, especially for more complicated proofs.

### 12.3.2 Exhaustive Input Testing

There are situations where the software component of an airborne system or equipment is simple and isolated such that the set of inputs and outputs can be bounded. If so, it may be possible to demonstrate that exhaustive testing of this input space can be substituted for a software verification process activity. For this alternative method, the applicant should include:

a. A complete definition of the set of valid inputs and outputs of the software.

b. An analysis which confirms the isolation of the inputs to the software.

c. Rationale for the exhaustive input test cases and procedures.

d. The test cases, test procedures and test results.

### 12.3.3 Considerations for Multiple-Version Dissimilar Software Verification

Guidelines follow concerning the software verification process as it applies to multiple-version dissimilar software. If the software verification process is modified because of the use of multiple-version dissimilar software, evidence should be provided that the software verification process objectives are satisfied and that equivalent error detection is achieved for each software version.

Multiple, dissimilar versions of the software are produced using combinations of these techniques:

- The Source Code is implemented in two or more different programming languages.
- The object code is generated using two or more different compilers.
- Each software version of Executable Object Code executes on a separate, dissimilar processor, or on a single processor with the means to provide partitioning between the software versions.
- The software requirements, software design, and/or Source Code are developed by two or more development teams whose interactions are managed.
- The software requirements, software design, and/or Source Code are developed on two or more software development environments, and/or each version is verified using separate test environments.
- The Executable Object Code is linked and loaded using two or more different linkage editors and two or more different loaders.

Ces critères peuvent être utilisés pour déterminer, au fur et à mesure du raffinement de la conception, quel sera l'ensemble des exigences auxquelles seront appliquées les méthodes formelles.

Degré de rigueur : Les méthodes formelles sont caractérisées par les niveaux de rigueur croissante suivants :

- Spécification formelle sans preuve.
- Spécification formelle avec preuves manuelles.
- Spécification formelle avec preuves contrôlées ou générées automatiquement.

L'utilisation de spécifications formelles seules rend les exigences non ambiguës. La preuve manuelle est un processus bien connu utilisable quand il y a peu de détail. Les preuves contrôlées ou générées automatiquement peuvent compléter les preuves manuelles et offrir un degré plus élevé de sûreté de fonctionnement, en particulier pour des preuves plus compliquées.

### 12.3.2 Test exhaustif des données d'entrée

Il y a des situations où le composant logiciel d'un système ou équipement de bord est suffisamment simple et isolé pour que l'on puisse borner l'ensemble des données d'entrée et de sortie. Dans ce cas, il est possible de démontrer que le test exhaustif du domaine d'entrée peut se substituer à une activité de vérification. Pour cette méthode de substitution, le postulant doit fournir :

a. Une définition complète de l'ensemble des données d'entrée et de sortie valides du logiciel.

b. Une analyse confirmant l'isolement des données d'entrée du logiciel.

c. La justification du caractère exhaustif des jeux et procédures de test.

d. Les jeux de test, procédures de test, et résultats de test.

### 12.3.3 Considérations relatives aux logiciels à versions multiples dissimilaires

Les recommandations ci-dessous concernent le processus de vérification du logiciel quand il s'applique à un logiciel à versions multiples dissimilaires. Si le processus de vérification du logiciel se trouve modifié en raison de l'utilisation d'un logiciel à versions multiples dissimilaires, on doit fournir la preuve que les objectifs de ce processus sont satisfaits et qu'une détection équivalente des erreurs est assurée pour chaque version du logiciel.

Les logiciels à versions multiples dissimilaires sont réalisés par combinaison des techniques suivantes :

- Le Code Source est écrit dans deux, ou plus de deux, langages de programmation différents.
- Le code objet est généré au moyen de deux, ou plus de deux, compilateurs différents.
- Chaque version du Code Objet Exécutable s'exécute sur un processeur distinct et dissimilaire, ou sur le même processeur mais avec un moyen de partitionnement entre les versions du logiciel.
- Les Spécifications du logiciel, sa conception, et/ou le Code Source sont développés par deux, ou plus de deux, équipes dont les interactions sont maîtrisées.
- Les Spécifications du logiciel, sa conception, et/ou le Code Source sont développés dans deux, ou plus de deux, environnements de développement, et/ou chaque version est vérifiée au moyen d'environnements de test différents.
- Deux ou plus de deux éditeurs de liens et chargeurs sont utilisés pour l'édition de lien et le chargement du Code Objet Exécutable.

- The software requirements, software design, and/or Source Code are developed in conformance with two or more different Software Requirements Standards, Software Design Standards, and/or Software Code Standards, respectively.

When multiple versions of software are used, the software verification methods may be modified from those used to verify single version software. They will apply to software development process activities that are multi-thread, such as separate, multiple development teams. The software verification process is dependent on the combined hardware and software architectures since this affects the dissimilarity of the multiple software versions. Additional software verification process objectives to be satisfied are:

a. To demonstrate that the inter-version compatibility requirements are satisfied, including compatibility during normal and abnormal operations and state transitions.

b. To demonstrate that equivalent error detection is achieved.

Other changes in software verification process activities may be agreed with by the certification authority, if the changes are substantiated by rationale that confirms equivalent software verification coverage.

12.3.3.1 Independence of Multiple-Version Dissimilar Software

When multiple-version dissimilar software versions are developed independently using a managed method, the development processes have the potential to reveal certain classes of errors such that verification of each software version is equivalent to independent verification of the software development processes. To realize the advantage of this potential, guidance for independence includes:

a. The applicant should demonstrate that different teams with limited interaction developed each software version's software requirements, software design and Source Code.

b. Independent test coverage analyses should still be performed as with a single version.

12.3.3.2 Multiple Processor-Related Verification

When each version of dissimilar software executes on a different type of processor, the verification of some aspects of compatibility of the code with the processor (paragraph 6.4.3, item a) may be replaced by verification to ensure that the multiple types of processor produce the correct outputs. This verification consists of integration tests in which the outputs of the multiple versions are cross-compared in requirements-based test cases. The applicant should show that:

a. Equivalent error detection is achieved.

b. Each processor was designed by a different developer.

c. The outputs of the multiple versions are equivalent.

12.3.3.3 Multiple-Version Source Code Verification

The guidelines for structural coverage analysis (subparagraph 6.4.4.2) may be modified for airborne systems or equipment using multiple-version dissimilar software. Structural coverage analysis may be performed at the Source Code level even if the object code is not directly traceable to Source Code statements provided that the applicant shows that:

a. Each version of software is coded using a different programming language.

b. Each compiler used is from a different developer.

- Les spécifications du logiciel, sa conception, et/ou le Code Source sont développés en conformité avec deux ou plus de deux règles de spécification, règles de conception, et/ou règles de codage différentes.

Lorsque des logiciels à versions multiples sont utilisés, les méthodes de vérification peuvent être adaptées à partir de celles applicables aux logiciels à version unique. Ces méthodes s'appliquent aux activités de développement qui sont dupliquées, comme par exemple plusieurs équipes de développement distinctes. Le processus de vérification dépend de la combinaison des architectures du matériel et du logiciel dans la mesure où celle-ci influe sur le caractère dissimilaire des versions. Les objectifs complémentaires à atteindre pour le processus de vérification du logiciel sont :

a. Démontrer que les exigences de compatibilité entre versions sont satisfaites, y compris la compatibilité en fonctionnement normal et anormal et pendant les changements d'états.

b. Démontrer qu'une détection équivalente des erreurs est assurée.

D'autres modifications des activités du processus de vérification peuvent être agréées par l'Autorité de certification, si ces modifications sont étayées par une justification démontrant une couverture équivalente de la vérification du logiciel.

12.3.3.1 Indépendance des logiciels à versions multiples dissimilaires

Lorsque des versions dissimilaires d'un logiciel sont développées de manière indépendante grâce à une méthode maîtrisée, les processus de développement permettent de mettre en évidence certaines classes d'erreurs de sorte que le processus de vérification de chaque version n'ait pas à satisfaire le critère d'indépendance. Pour tirer avantage de cette potentialité, les directives relatives à l'indépendance sont :

a. Le postulant doit démontrer que les spécifications du logiciel, la conception du logiciel, et le Code Source de chaque version du logiciel ont été développés par des équipes différentes ayant entre elles des interactions limitées.

b. Les analyses de couverture de tests doivent, comme pour une version unique, être menées de manière indépendante.

12.3.3.2 Vérifications relatives aux processeurs multiples

Quand chaque version d'un logiciel dissimilaire s'exécute sur un type différent de processeur, on peut remplacer la vérification de certains aspects de la compatibilité du code et du processeur (paragraphe 6.4.3 a) par la vérification que les différents types de processeur génèrent les sorties correctes. Cette vérification est constituée par des tests d'intégration dans lesquels est effectuée une comparaison croisée des sorties des versions multiples avec des jeux de test fonctionnels. Le postulant doit montrer que :

a. Une détection équivalente des erreurs est assurée.

b. Chaque processeur a été conçu par un concepteur différent.

c. Les sorties des versions multiples sont équivalentes.

12.3.3.3 Vérifications du Code Source des versions multiples

Les recommandations liées à l'analyse de la couverture structurelle (sous-paragraphe 6.4.4.2) pour les systèmes ou équipements de bord utilisant du logiciel dissimilaire peuvent être adaptées. L'analyse de couverture structurelle peut être effectuée sur le Code Source même si le code objet n'est pas directement traçable vers les instructions du Code Source à condition que le postulant prouve que :

a. Chaque version du logiciel est codée au moyen d'un langage de programmation différent.

b. Chaque compilateur utilisé provient d'une souche de conception différente.

12.3.3.4 Tool Qualification for Multiple-Version Dissimilar Software

If multiple-version dissimilar software is used, the tool qualification process may be modified, if evidence is available that the multiple software development tools are dissimilar. This depends on the demonstration of equivalent software verification process activity in the development of the multiple software versions using dissimilar software development tools. The applicant should show that:

a. Each tool was obtained from a different developer.

b. Each tool has a dissimilar design.

12.3.3.5 Multiple Simulators and Verification

If separate, dissimilar simulators are used to verify multiple-version dissimilar software versions, then tool qualification of the simulators may be modified. This depends on the demonstration of equivalent software verification process activity in the simulation of the multiple software versions using multiple simulators. Unless it can be justified as unnecessary, for multiple simulators to be dissimilar, evidence should be available that:

a. Each simulator was developed by a different team.

b. Each simulator has different requirements, a different design and a different programming language.

c. Each simulator executes on a different processor.

*NOTE: When a multiple processor system using multiple, dissimilar versions of software are executing on identical processors, it may be difficult to demonstrate dissimilarity of simulators because of the reliance on information obtained from a common source, the processor manufacturer.*

12.3.4 Software Reliability Models

During the preparation of this document, methods for estimating the post-verification probabilities of software errors were examined. The goal was to develop numerical requirements for such probabilities for software in computer-based airborne systems or equipment. The conclusion reached, however, was that currently available methods do not provide results in which confidence can be placed to the level required for this purpose. Hence, this document does not provide guidance for software error rates. If the applicant proposes to use software reliability models for certification credit, rationale for the model should be included in the Plan for Software Aspects of Certification, and agreed with by the certification authority.

12.3.5 Product Service History

If equivalent safety for the software can be demonstrated by the use of the software's product service history, some certification credit may be granted. The acceptability of this method is dependent on:

- Configuration management of the software.
- Effectiveness of problem reporting activity.
- Stability and maturity of the software.
- Relevance of product service history environment.
- Actual error rates and product service history.
- Impact of modifications.

12.3.3.4 Qualification d'outil pour les logiciels à versions multiples dissimilaires

Dans le cas d'utilisation de logiciel à versions multiples dissimilaires, le processus de qualification d'outil peut être adapté si on a la preuve que les outils de développement sont dissimilaires. Ceci dépend de la démonstration que les activités de vérification des versions multiples au moyen d'outils de développement dissimilaires sont équivalentes (cf. Paragraphe 12.2.1). Le postulant doit démontrer que :

a. Chaque outil provient d'un concepteur différent.

b. Chaque outil possède une conception dissimilaire.

12.3.3.5 Simulateurs multiples et vérification

Si des simulateurs dissimilaires et séparés sont utilisés pour vérifier les versions d'un logiciel à versions multiples dissimilaires, la qualification des simulateurs peut être modifiée. Cela dépend de la démonstration que, lors de l'activité du processus de vérification du logiciel, les simulations des différentes versions, avec des simulateurs multiples sont équivalentes(Cf Paragraphe 12.2.2). Pour démontrer que des simulateurs multiples sont dissimilaires, le postulant doit apporter (sauf si justification peut être faite que ce n'est pas nécessaire) la preuve que:

a. Chaque simulateur a été développé par une équipe différente.

b. Chaque simulateur est soumis à des exigences différentes, et possède une conception et un langage de programmation différents.

c. Chaque simulateur agit sur un processeur différent.

*NOTA :* *Quand un système à processeurs multiples utilise des versions dissimilaires agissant sur des processeurs identiques, il peut être difficile de démontrer le caractère dissimilaire des simulateurs en raison de la confiance dans les informations obtenues d'une source commune, le constructeur du processeur.*

12.3.4 Modèles de fiabilité de logiciel

Au cours de la préparation de ce document, des méthodes d'estimation de probabilité d'erreurs de logiciel résiduelles après vérification ont été examinées. Le but était d'aboutir à des exigences chiffrées de ces probabilités pour les systèmes et équipements de bord. On est cependant arrivé à la conclusion que les méthodes actuelles ne donnent pas de résultats suffisamment fiables pour les niveaux de sécurité requis. C'est pourquoi, ce document ne donne pas de directives pour les taux d'erreur de logiciel. Si le postulant propose d'utiliser des modèles de fiabilité de logiciel pour obtenir un crédit de certification, la justification du modèle doit être indiquée dans le Plan des Aspects Logiciels de la Certification et agréée par l'Autorité de certification.

12.3.5 Historique du produit en service

Un certain crédit de certification peut être accordé s'il est possible de démontrer un niveau de sécurité équivalent en utilisant l'historique du produit en service. L'acceptabilité de cette méthode dépend de :

- La gestion de configuration du logiciel.
- L'efficience de l'activité de compte rendu des anomalies.
- La stabilité et la maturité du logiciel.
- La représentativité de l'environnement de l'historique du produit en service.
- Les taux d'erreurs constatés et l'historique du produit en service.
- L'impact des modifications.

Guidance for the use of product service history includes:

a. The applicant should show that the software and associated evidence used to comply with system safety objectives have been under configuration management throughout the product service history.

b. The applicant should show that the problem reporting during the product service history provides assurance that representative data is available and that in-service problems were reported and recorded, and are retrievable.

c. Configuration changes during the product service history should be identified and the effect analyzed to confirm the stability and maturity of the software. Uncontrolled changes to the Executable Object Code during the product service history may invalidate the use of product service history.

d. The intended software usage should be analyzed to show the relevance of the product service history.

e. If the operating environments of the existing and proposed applications differ, additional software verification should confirm compliance with the system safety objectives

f. The analysis of configuration changes and product service history environment may require the use of software requirements and design data to confirm the applicability of the product service history environment.

g. If the software is a subset of the software that was active during the service period, then analysis should confirm the equivalency of the new environment with the previous environment, and determine those software components that were not executed during normal operation.

   *NOTE: Additional verification may be needed to confirm compliance with the system safety objectives for those components.*

h. The problem report history should be analyzed to determine how safety-related problems occurred and which problems were corrected.

i. Those problems that are indicative of an inadequate process, such as design or code errors, should be indicated separately from those whose cause are outside the scope of this document, such as hardware or system requirements errors.

j. The data described above and these items should be specified in the Plan for Software Aspects of Certification:

   (1) Analysis of the relevance of the product service history environment.

   (2) Length of service period and rationale for calculating the number of hours in service, including factors such as operational modes, the number of independently operating copies in the installation and in service, and the definition of "normal operation" and "normal operation time."

   (3) Definition of what was counted as an error and rationale for that definition.

   (4) Proposed acceptable error rates and rationale for the product service history period in relation to the system safety and proposed error rates.

k. If the error rate is greater than that identified in the plan, these errors should be analyzed and the analyses reviewed with the certification authority.

Les directives pour l'utilisation de l'historique du produit en service comprennent en particulier :

a. Le postulant doit démontrer que le logiciel et les preuves associées utilisées pour la conformité aux objectifs de sécurité ont été gérés en configuration pendant tout l'historique du produit en service.

b. Le postulant doit démontrer que le compte rendu des anomalies survenues au cours de l'historique du produit en service garantit qu'il existe des données représentatives, que les problèmes rencontrés en service ont été consignés et enregistrés, et que ces informations peuvent être exploitées.

c. Les changements de configuration survenus au cours de l'historique du produit en service doivent être identifiés et leur effet analysé afin de confirmer la stabilité et la maturité du logiciel. Des modifications non contrôlées du Code Objet Exécutable pendant l'historique du produit en service peuvent invalider l'utilisation de cet historique.

d. L'utilisation prévue du logiciel doit être analysée afin de démontrer la pertinence de l'historique du produit en service.

e. Si les environnements de fonctionnement des applications existantes et de celles proposées sont différents, des vérifications complémentaires du logiciel doivent confirmer la conformité aux objectifs de sécurité du système.

f. L'analyse des changements de configuration et celle de l'environnement au cours de l'historique du produit en service peuvent imposer d'utiliser les spécifications du logiciel ainsi que les données de conception, afin de confirmer l'applicabilité de cet environnement.

g. Si le logiciel est un sous-ensemble du logiciel utilisé pendant l'historique en service, l'analyse doit alors confirmer l'équivalence du nouvel environnement avec l'environnement antérieur et déterminer quels composants de ce logiciel n'ont pas été exécutés en fonctionnement normal.

   *NOTA :* *Des vérifications complémentaires de ces composants peuvent être nécessaires afin de confirmer la conformité aux objectifs de sécurité du système.*

h. L'historique des rapports d'anomalies doit être analysé afin de déterminer comment les problèmes liés à la sécurité ont pu se produire et lesquels ont été corrigés.

i. Les problèmes révélant un processus inadéquat, tels que les erreurs de conception ou de codage, doivent être identifiés séparément de ceux dont la cause est hors du domaine couvert par ce document, tels que les erreurs liées au matériel ou aux spécifications du système.

j. Les données décrites ci-dessus ainsi que les points suivants doivent être précisés dans le Plan des Aspects Logiciels de la Certification :

   (1) L'analyse de la représentativité de l'environnement au cours de l'historique du produit en service.

   (2) La durée de la période de service et les justifications pour le calcul du nombre d'heures en service, y compris des facteurs comme les modes de fonctionnement, le nombre de copies installées et en service fonctionnant indépendamment, et la définition du "fonctionnement normal" et du "temps de fonctionnement normal".

   (3) La définition de ce qui est comptabilisé comme une erreur et la justification de cette définition.

   (4) La proposition, en relation avec la sécurité du système, des taux d'erreur acceptables et de leur justification sur la période de l'historique du produit en service, et la proposition des taux d'erreur observés.

k. Si le taux d'erreur est supérieur à celui identifié dans le plan, ces erreurs doivent être analysées et ces analyses examinées avec l'Autorité de certification.

# ANNEX A

## PROCESS OBJECTIVES AND OUTPUTS BY SOFTWARE LEVEL

This annex provides guidelines for the software life cycle process objectives and outputs described in this document by software level. These tables reference the objectives and outputs of the software life cycle processes previously described in this document. The tables include guidelines for:

a. The process objectives applicable for each software level. For level E software, see paragraph 2.2.2.

b. The independence by software level of the software life cycle process activities applicable to satisfy that process's objectives.

c. The control category by software level for the software life cycle data produced by the software life cycle process activities (subsection 7.3).

# ANNEXE A

## OBJECTIFS ET PRODUITS DES PROCESSUS PAR NIVEAU LOGICIEL

Cette annexe fournit des recommandations concernant les objectifs et les produits des processus de cycle de vie du logiciel, décrits dans ce document par niveau logiciel. Ces tableaux référencent les objectifs et les produits des processus du cycle de vie du logiciel décrits antérieurement dans ce document.

Ces tableaux comprennent des recommandations pour :

a. Les objectifs de processus applicables pour chaque niveau logiciel. Pour le niveau logiciel E, voir le paragraphe 2.2.2.

b. L'indépendance par niveau logiciel des activités des processus du cycle de vie du logiciel applicables pour satisfaire les objectifs de ces processus.

c. La catégorie de contrôle par niveau logiciel pour les données du cycle de vie du logiciel générées par les activités des processus (sous-section 7.3).

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Software development and integral processes activities are defined. | 4.1a<br>4.3 | ❍ | ❍ | ❍ | ❍ | Plan for Software Aspects of Certification<br>Software Development Plan<br>Software Verification Plan<br>SCM Plan<br>SQA Plan | 11.1<br>11.2<br>11.3<br>11.4<br>11.5 | ①<br>①<br>①<br>①<br>① | ①<br>①<br>①<br>①<br>① | ①<br>②<br>②<br>②<br>② | ①<br>②<br>②<br>②<br>② |
| 2 | Transition criteria, inter-relationships and sequencing among processes are defined. | 4.1b<br>4.3 | ❍ | ❍ | ❍ | | | | | | | |
| 3 | Software life cycle environment is defined. | 4.1c | ❍ | ❍ | ❍ | | | | | | | |
| 4 | Additional considerations are addressed. | 4.1d | ❍ | ❍ | ❍ | ❍ | | | | | | |
| 5 | Software development standards are defined. | 4.1e | ❍ | ❍ | ❍ | | SW Requirements Standards<br>SW Design Standards<br>SW Code Standards | 11.6<br>11.7<br>11.8 | ①<br>①<br>① | ①<br>①<br>① | ②<br>②<br>② | |
| 6 | Software plans comply with this document. | 4.1f<br>4.6 | ❍ | ❍ | ❍ | | SQA Records<br>Software Verification Results | 11.19<br>11.14 | ②<br>② | ②<br>② | ②<br>② | |
| 7 | Software plans are coordinated. | 4.1g<br>4.6 | ❍ | ❍ | ❍ | | SQA Records<br>Software Verification Results | 11.19<br>11.14 | ②<br>② | ②<br>② | ②<br>② | |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-1: SOFTWARE PLANNING PROCESS

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Les activités de développement du logiciel et des processus intégraux sont définies. | 4.1a<br>4.3 | ❍ | ❍ | ❍ | ❍ | Plan pour les aspects logiciels de la certification<br>Plan de développement logiciel<br>Plan de vérification logicielle<br>Plan SCM<br>Plan SQA | 11.1<br>11.2<br>11.3<br>11.4<br>11.5 | ①<br>①<br>①<br>①<br>① | ①<br>①<br>①<br>①<br>① | ①<br>②<br>②<br>②<br>② | ①<br>②<br>②<br>②<br>② |
| 2 | Les critères de transition, les relations réciproque et les enchaînements des processus sont définis. | 4.1b<br>4.3 | ❍ | ❍ | ❍ | | | | | | | |
| 3 | L'environnement du cycle de vie du logiciel est défini. | 4.1c | ❍ | ❍ | ❍ | | | | | | | |
| 4 | Les considérations complémentaires sont prises en compte. | 4.1d | ❍ | ❍ | ❍ | ❍ | | | | | | |
| 5 | Les normes de développement logiciel sont définies. | 4.1e | ❍ | ❍ | ❍ | | Normes d'exigences logicielles<br>Normes de conception logicielle<br>Normes de code logiciels | 11.6<br>11.7<br>11.8 | ①<br>①<br>① | ①<br>①<br>① | ②<br>②<br>② | |
| 6 | Les plans logiciels sont conformes à ce document. | 4.1f<br>4.6 | ❍ | ❍ | ❍ | | Enregistrement SQA<br>Résultats des vérif. logicielles | 11.19<br>11.14 | ②<br>② | ②<br>② | ②<br>② | |
| 7 | Les plans logiciels sont coordonnées. | 4.1g<br>4.6 | ❍ | ❍ | ❍ | | Enregistrement SQA<br>Résultats des vérif. logicielles | 11.19<br>11.14 | ②<br>② | ②<br>② | ②<br>② | |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-1 : PLANIFICATION DU LOGICIEL

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | High-level requirements are developed. | 5.1.1a | ❍ | ❍ | ❍ | ❍ | Software Requirements Data | 11.9 | ① | ① | ① | ① |
| 2 | Derived high-level requirements are defined. | 5.1.1b | ❍ | ❍ | ❍ | ❍ | Software Requirements Data | 11.9 | ① | ① | ① | ① |
| 3 | Software architecture is developed. | 5.2.1a | ❍ | ❍ | ❍ | ❍ | Design Description | 11.10 | ① | ① | ② | ② |
| 4 | Low-level requirements are developed. | 5.2.1a | ❍ | ❍ | ❍ | ❍ | Design Description | 11.10 | ① | ① | ② | ② |
| 5 | Derived low-level requirements are defined. | 5.2.1b | ❍ | ❍ | ❍ | ❍ | Design Description | 11.10 | ① | ① | ② | ② |
| 6 | Source Code is developed. | 5.3.1a | ❍ | ❍ | ❍ | ❍ | Source Code | 11.11 | ① | ① | ① | ① |
| 7 | Executable Object Code is produced and integrated in the target computer. | 5.4.1a | ❍ | ❍ | ❍ | ❍ | Executable Object Code | 11.12 | ① | ① | ① | ① |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-2: SOFTWARE DEVELOPMENT PROCESSES

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Les exigences de haut niveau sont développées. | 5.1.1a | ❍ | ❍ | ❍ | ❍ | Description de Spécification | 11.9 | ① | ① | ① | ① |
| 2 | Les exigences de haut niveau dérivées sont définies. | 5.1.1b | ❍ | ❍ | ❍ | ❍ | Description de Spécification | 11.9 | ① | ① | ① | ① |
| 3 | L'architecture du logiciel est développée. | 5.2.1a | ❍ | ❍ | ❍ | ❍ | Description de Conception | 11.10 | ① | ① | ② | ② |
| 4 | Les exigences de bas niveau sont développées. | 5.2.1a | ❍ | ❍ | ❍ | ❍ | Description de Conception | 11.10 | ① | ① | ② | ② |
| 5 | Les exigences de bas niveau dérivées sont définies. | 5.2.1b | ❍ | ❍ | ❍ | ❍ | Description de Conception | 11.10 | ① | ① | ② | ② |
| 6 | Le Code Source est développé. | 5.3.1a | ❍ | ❍ | ❍ | ❍ | Code Source | 11.11 | ① | ① | ① | ① |
| 7 | Le Code Objet Exécutable est produit et intégré dans la machine cible. | 5.4.1a | ❍ | ❍ | ❍ | ❍ | Code Objet Exécutable | 11.12 | ① | ① | ① | ① |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-2 : DEVELOPPEMENT DU LOGICIEL

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Software high-level requirements comply with system requirements. | 6.3.1a | ● | ● | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 2 | High-level requirements are accurate and consistent. | 6.3.1b | ● | ● | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 3 | High-level requirements are compatible with target computer. | 6.3.1c | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 4 | High-level requirements are verifiable. | 6.3.1d | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 5 | High-level requirements conform to standards. | 6.3.1e | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 6 | High-level requirements are traceable to system requirements. | 6.3.1f | ❍ | ❍ | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 7 | Algorithms are accurate. | 6.3.1g | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-3: VERIFICATION OF OUTPUTS OF SOFTWARE REQUIREMENTS PROCESS

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Les exigences de haut niveau sont en conformité avec les spécifications du système. | 6.3.1a | ● | ● | ❍ | ❍ | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | ② |
| 2 | Les exigences de haut niveau sont précises et cohérentes. | 6.3.1b | ● | ● | ❍ | ❍ | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | ② |
| 3 | Les exigences de haut niveau sont compatibles avec la machine cible. | 6.3.1c | ❍ | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 4 | Les exigences de haut niveau sont vérifiables. | 6.3.1d | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 5 | Les exigences de haut niveau sont en conformité avec les règles. | 6.3.1e | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 6 | Les exigences de haut niveau sont traçables vers les spécifications au système. | 6.3.1f | ❍ | ❍ | ❍ | ❍ | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | ② |
| 7 | Les algorithmes sont précis. | 6.3.1g | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-3 : VERIFICATION DES PRODUITS DES SPECIFICATIONS DU LOGICIEL

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Low-level requirements comply with high-level requirements. | 6.3.2a | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 2 | Low-level requirements are accurate and consistent. | 6.3.2b | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 3 | Low-level requirements are compatible with target computer. | 6.3.2c | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 4 | Low-level requirements are verifiable. | 6.3.2d | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 5 | Low-level requirements conform to standards. | 6.3.2e | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 6 | Low-level requirements are traceable to high-level requirements. | 6.3.2f | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 7 | Algorithms are accurate. | 6.3.2g | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 8 | Software architecture is compatible with high-level requirements. | 6.3.3a | ● | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 9 | Software architecture consistent. | 6.3.3b | ● | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 10 | Software architecture is compatible with target computer. | 6.3.3c | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 11 | Software architecture is verifiable. | 6.3.3d | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 12 | Software architecture conforms to standards. | 6.3.3e | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 13 | Software partitioning integrity is confirmed. | 6.3.3f | ● | ❍ | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-4: VERIFICATION OF OUTPUTS OF SOFTWARE DESIGN PROCESS

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Les exigences de bas niveau sont en conformité avec les exigences de haut niveau. | 6.3.2a | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 2 | Les exigences de bas niveau sont précises et cohérentes. | 6.3.2b | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 3 | Les exigences de bas niveau sont compatibles avec la machine cible. | 6.3.2c | ❍ | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 4 | Les exigences de bas niveau sont vérifiables. | 6.3.2d | ❍ | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 5 | Les exigences de bas niveau sont conformes aux règles. | 6.3.2e | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 6 | Les exigences de bas niveau sont traçables vers les exigences de haut niveau. | 6.3.2f | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 7 | Les algorithmes sont précis. | 6.3.2g | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 8 | L'architecture du logiciel est compatible avec les exigences de haut niveau. | 6.3.3a | ● | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 9 | L'architecture du logiciel est cohérente. | 6.3.3b | ● | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 10 | L'architecture du logiciel est compatible avec la machine cible. | 6.3.3c | ❍ | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 11 | L'architecture du logiciel est vérifiable. | 6.3.3d | ❍ | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 12 | L'architecture du logiciel est conforme aux règles. | 6.3.3e | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 13 | L'intégrité du partitionnement est confirmée. | 6.3.3f | ● | ❍ | ❍ | ❍ | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | ② |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-4 : VERIFICATION DES PRODUITS DE LA CONCEPTION LOGICIEL

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Source Code complies with low-level requirements. | 6.3.4a | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 2 | Source Code complies with software architecture. | 6.3.4b | ● | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 3 | Source Code is verifiable. | 6.3.4c | ❍ | ❍ | | | Software Verification Results | 11.14 | ② | ② | | |
| 4 | Source Code conforms to standards. | 6.3.4d | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 5 | Source Code is traceable to low-level requirements. | 6.3.4e | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 6 | Source Code is accurate and consistent. | 6.3.4f | ● | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 7 | Output of software integration process is complete and correct. | 6.3.5 | ❍ | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-5: VERIFICATION OF OUTPUT SOFTWARE CODING AND INTEGRATION PROCESSES

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de Contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Le Code Source est en conformité avec les exigences de bas niveau. | 6.3.4a | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 2 | Le Code Source est conforme à l'architecture du logiciel. | 6.3.4b | ● | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 3 | Le Code Source est vérifiable. | 6.3.4c | ❍ | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 4 | Le Code Source est conforme aux règles. | 6.3.4d | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 5 | Le Code Source est traçable vers les exigences de bas niveau. | 6.3.4e | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 6 | Le Code Source est précis et cohérent. | 6.3.4f | ● | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 7 | La sortie du processus d'intégration du logiciel est complète et correcte. | 6.3.5 | ❍ | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-5 : VERIFICATION DES PRODUITS DU CODAGE ET DE L'INTEGRATION DU LOGICIEL

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Executable Objet Code complies with high-level requirements. | 6.4.2.1<br>6.4.3 | ❍ | ❍ | ❍ | ❍ | Software Verification Cases And Procedures.<br>Software Verification Results | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | ②<br>② |
| 2 | Executable Object Code is robust with high-level requirements. | 6.4.2.2<br>6.4.3 | ❍ | ❍ | ❍ | ❍ | Software Verification Cases And Procedures.<br>Software Verification Results | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | ②<br>② |
| 3 | Executable Object Code complies with low-level requirements. | 6.4.2.1<br>6.4.3 | ● | ● | ❍ | | Software Verification Cases And Procedures.<br>Software Verification Results | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | |
| 4 | Executable Object Code is robust with low-level requirements. | 6.4.2.2<br>6.4.3 | ● | ❍ | ❍ | | Software Verification Cases And Procedures.<br>Software Verification Results | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | |
| 5 | Executable Object Code is compatible with target computer. | 6.4.3a | ❍ | ❍ | ❍ | ❍ | Software Verification Cases And Procedures.<br>Software Verification Results | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | ②<br>② |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-6: TESTING OF OUTPUTS OF INTEGRATION PROCESS

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Le Code Objet Exécutable est conforme aux exigences de haut niveau. | 6.4.2.1<br>6.4.3 | ❍ | ❍ | ❍ | ❍ | Jeux et Procédures de Vérification du Logiciel<br>Résultats de Vérification du Logiciel | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | ②<br>② |
| 2 | Le Code Objet Exécutable est robuste vis-à-vis des exigences de haut niveau. | 6.4.2.2<br>6.4.3 | ❍ | ❍ | ❍ | ❍ | Jeux et Procédures de Vérification du Logiciel<br>Résultats de Vérification du Logiciel | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | ②<br>② |
| 3 | Le Code Objet Exécutable est conforme aux exigences de bas niveau. | 6.4.2.1<br>6.4.3 | ● | ● | ❍ | | Jeux et Procédures de Vérification du Logiciel<br>Résultats de Vérification du Logiciel | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | |
| 4 | Le Code Objet Exécutable est robuste vis-à-vis des exigences de bas niveau. | 6.4.2.2<br>6.4.3 | ● | ❍ | ❍ | | Jeux et Procédures de Vérification du Logiciel<br>Résultats de Vérification du Logiciel | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | |
| 5 | Le Code Objet Exécutable est compatible avec la machine cible. | 6.4.3a | ❍ | ❍ | ❍ | ❍ | Jeux et Procédures de Vérification du Logiciel<br>Résultats de Vérification du Logiciel | 11.13<br>11.14 | ①<br>② | ①<br>② | ②<br>② | ②<br>② |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-6 : TEST DES PRODUITS DE L'INTEGRATION

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Test procedures are correct. | 6.3.6b | ● | ❍ | ❍ | | Software Verification Cases and Procedures | 11.13 | ② | ② | ② | |
| 2 | Test results are correct and discrepancies explained. | 6.3.6c | ● | ❍ | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 3 | Test coverage of high-level requirements is achieved. | 6.4.4.1 | ● | ❍ | ❍ | ❍ | Software Verification Results | 11.14 | ② | ② | ② | ② |
| 4 | Test coverage of low-level requirements is achieved. | 6.4.4.1 | ● | ❍ | | | Software Verification Results | 11.14 | ② | ② | ② | |
| 5 | Test coverage of software structure (modified condition/decision) is achieved. | 6.4.4.2 | ● | | | | Software Verification Results | 11.14 | ② | | | |
| 6 | Test coverage of software structure (decision coverage) is achieved. | 6.4.4.2a<br>6.4.4.2b | ● | ● | | | Software Verification Results | 11.14 | ② | ② | | |
| 7 | Test coverage of software structure (statement coverage) is achieved. | 6.4.4.2a<br>6.4.4.2b | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |
| 8 | Test coverage of software structure (data coupling and control coupling) is achieved. | 6.4.4.2c | ● | ● | ❍ | | Software Verification Results | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-7: VERIFICATION OF VERIFICATION PROCESS RESULTS

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Les procédures de test sont correctes. | 6.3.6b | ● | ❍ | ❍ | | Jeux et Procédures de Vérification du Logiciel | 11.13 | ② | ② | ② | |
| 2 | Les résultats des tests sont correctes et les écarts sont expliqués. | 6.3.6c | ● | ❍ | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 3 | La couverture de test des exigences de haut niveau est assurée. | 6.4.4.1 | ● | ❍ | ❍ | ❍ | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | ② |
| 4 | La couverture de test de bas niveau est assurée. | 6.4.4.1 | ● | ❍ | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 5 | La couverture structurelle (condition/décision modifiée) est assurée. | 6.4.4.2 | ● | | | | Résultats de Vérification du Logiciel | 11.14 | ② | | | |
| 6 | La couverture structurelle (décision) est assurée. | 6.4.4.2a<br>6.4.4.2b | ● | ● | | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | | |
| 7 | La couverture structurelle (instruction) est assurée. | 6.4.4.2a<br>6.4.4.2b | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |
| 8 | La couverture structurelle (couplage de données et couplage de contrôle) est assurée. | 6.4.4.2c | ● | ● | ❍ | | Résultats de Vérification du Logiciel | 11.14 | ② | ② | ② | |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-7 : VERIFICATION DES PRODUITS DE LA VERIFICATION

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Configuration items are identified. | 7.2.1 | ❍ | ❍ | ❍ | ❍ | SCM Records | 11.18 | ② | ② | ② | ② |
| 2 | Baselines and traceability are established. | 7.2.2 | ❍ | ❍ | ❍ | | Software Configuration Index<br>SCM Records | 11.16<br>11.18 | ①<br>② | ①<br>② | ①<br>② | ①<br>② |
| 3 | Problem reporting, change control, change review, and configuration status accounting are established. | 7.2.3<br>7.2.4<br>7.2.5<br>7.2.6 | ❍ | ❍ | ❍ | ❍ | Problem Reports<br>SCM Records | 11.17<br>11.18 | ②<br>② | ②<br>② | ②<br>② | ②<br>② |
| 4 | Archive, retrieval, and release are established. | 7.2.7 | ❍ | ❍ | ❍ | ❍ | SCM Records | 11.18 | ② | ② | ② | ② |
| 5 | Software load control is established. | 7.2.8 | ❍ | ❍ | ❍ | ❍ | SCM Records | 11.18 | ② | ② | ② | ② |
| 6 | Software life cycle environment control is established. | 7.2.9 | ❍ | ❍ | ❍ | ❍ | Software Life Cycle Environment Configuration Index<br>SCM Records | 11.15<br>11.18 | ①<br>② | ①<br>② | ①<br>② | ②<br>② |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-8: SOFTWARE CONFIGURATION MANAGMENT PROCESS

*NOTE (1): Although the software configuration management objectives of section 7 do not vary with software level, the control category assigned to the software life cycle data may vary.*

*NOTE (2): The objectives of section 7 provide a sufficient integrity basis in the SCM process activities without the need for the independence criteria.*

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Les éléments de configuration sont identifiés. | 7.2.1 | ❍ | ❍ | ❍ | ❍ | Documents SCM | 11.18 | ② | ② | ② | ② |
| 2 | Les référentiels et la traçabilité sont établis. | 7.2.2 | ❍ | ❍ | ❍ | | Répertoire de Configuration du Logiciel<br>Documents SCM | 11.15<br>11.18 | ①<br>② | ①<br>② | ①<br>② | ①<br>② |
| 3 | Le compte-rendu des anomalies, la gestion des modifications, la revue des modifications et le suivi des états de configuration sont établis. | 7.2.3<br>7.2.4<br>7.2.5<br>7.2.6 | ❍ | ❍ | ❍ | ❍ | Rapports d'anomalies<br>Documents SCM | 11.17<br>11.18 | ②<br>② | ②<br>② | ②<br>② | ②<br>② |
| 4 | L'archivage, la restauration et la mise à disposition sont établis. | 7.2.7 | ❍ | ❍ | ❍ | ❍ | Documents SCM | 11.18 | ② | ② | ② | ② |
| 5 | Le contrôle du chargement du logiciel est établi. | 7.2.8 | ❍ | ❍ | ❍ | ❍ | Documents SCM | 11.18 | ② | ② | ② | ② |
| 6 | Le contrôle de l'environnement du cycle de vie du logiciel est établi. | 7.2.9 | ❍ | ❍ | ❍ | ❍ | Répertoire de la Configuration de l'Environnement du Logiciel<br>Documents SCM | 11.15<br>11.18 | ①<br>② | ①<br>② | ①<br>② | ②<br>② |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-8 : GESTION DE CONFIGURATION

*NOTA 1 : Bien que les objectifs de gestion de configuration du logiciel de la section 7 soient indépendants du niveau logiciel, la catégorie de contrôle attribuée aux données du cycle de vie du logiciel peut varier.*

*NOTA 2 : Les objectifs de la section 7 fournissent une base d'intégrité suffisante pour les activités du processus de SCM sans qu'il y ait de besoin de critères.*

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Assurance is obtained that software development and integral processes comply with approved software plans and standards. | 8.1a | ● | ● | ● | ● | Software Quality Assurance (SQA) Records | 11.19 | ② | ② | ② | ② |
| 2 | Assurance is obtained that transition criteria for the software life cycle processes are satisfied. | 8.1b | ● | ● | | | SQA Records | 11.19 | ② | ② | | |
| 3 | Software conformity review is conducted. | 8.1c<br>8.3 | ● | ● | ● | ● | SQA Records | 11.19 | ② | ② | ② | ② |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-9: SOFTWARE QUALITY ASSURANCE PROCESS

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | L'assurance est acquise que le développement du logiciel et les processus intégraux sont en conformité avec les règles et les plans. | 8.1a | ● | ● | ● | ● | Documents d'Assurance Qualité du Logiciel (SQA) | 11.19 | ② | ② | ② | ② |
| 2 | L'assurance est acquise que les critères de transition pour le cycle de vie du logiciel sont satisfaits. | 8.1b | ● | ● | | | Documents d'Assurance Qualité du Logiciel (SQA) | 11.19 | ② | ② | | |
| 3 | La revue de conformité est menée. | 8.1c<br>8.3 | ● | ● | ● | ● | Documents d'Assurance Qualité du Logiciel (SQA) | 11.19 | ② | ② | ② | ② |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-9 : ASSURANCE QUALITE

| | Objective | | Applicability by SW level | | | | Output | | Control category by SW level | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Ref.** | **A** | **B** | **C** | **D** | **Description** | **Ref.** | **A** | **B** | **C** | **D** |
| 1 | Communication and understand between the applicant and the certification authority is established. | 9.0 | ❍ | ❍ | ❍ | ❍ | Plan for Software Aspects of Certification | 11.1 | ① | ① | ① | ① |
| 2 | The means of compliance is proposed and agreement with the Plan for Software Aspect of Certification is obtained. | 9.1 | ❍ | ❍ | ❍ | ❍ | Plan for Software Aspect Certification | 11.1 | ① | ① | ① | ① |
| 3 | Compliance substantiation is provided. | 9.2 | ❍ | ❍ | ❍ | ❍ | Software Accomplishment Summary<br>Software Configuration Index | 11.20<br>11.16 | ①<br>① | ①<br>① | ①<br>① | ①<br>① |

| | | |
|---|---|---|
| LEGEND | ● | The objective should be satisfied with independence. |
| | ❍ | The objective should be satisfied. |
| | Blank | Satisfaction of objective is at applicant's discretion. |
| | ① | Data satisfies the objectives of Control Category 1 (CC1). |
| | ② | Data satisfies the objectives of Control Category 2 (CC2). |

TABLE A-10: CERTIFICATION LIAISON PROCESS

*NOTE: The plan for software aspects of certification and the software Configuration Index are outputs to other processes. They are included in this table to show completion of the certification liaison process objectives .*

| | Objectif | | Applicabilité par niveau logiciel | | | | Produit | | Catégorie de contrôle par niveau logiciel | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Description** | **Réf.** | **A** | **B** | **C** | **D** | **Description** | **Réf.** | **A** | **B** | **C** | **D** |
| 1 | Communication et compréhension entre le postulant et l'Autorité de certification sont établies. | 9.0 | ❍ | ❍ | ❍ | ❍ | Plan des Aspects Logiciels de la Certification | 11.1 | ① | ① | ① | ① |
| 2 | Les moyens de conformité sont proposés et le satisfaction du Plan des Aspects Logiciels de la Certification est obtenue. | 9.1 | ❍ | ❍ | ❍ | ❍ | Plan des Aspects Logiciels de la Certification | 11.1 | ① | ① | ① | ① |
| 3 | Justification de conformité fournie. | 9.2 | ❍ | ❍ | ❍ | ❍ | Résumé des Travaux Réalisés<br>Répertoire de la Configuration du Logiciel | 11.20<br>11.16 | ①<br>① | ①<br>① | ①<br>① | ①<br>① |

| | | |
|---|---|---|
| LEGENDE | ● | L'objectif doit être atteint avec "indépendance". (*) |
| | ❍ | L'objectif doit être atteint. |
| | Blanc | La réalisation de l'objectif est laissée à la discrétion du postulant. |
| | ① | Les données sont conformes à la Catégorie de Contrôle 1 (CC1). |
| | ② | Les données sont conformes à la Catégorie de Contrôle 2 (CC2). |
| | (*) | Voir glossaire |

TABLEAU A-10 : COORDINATION POUR LA CERTIFICATION

*NOTA : Le Plan des Aspects Logiciels de la Certification et le Répertoire de la Configuration du Logiciel sont des produits d'autre processus. Ils sont inclus dans ce tableau pour démontrer la réalisation des objectifs du processus de coordination pour la certification.*

# ANNEX B

## ACRONYMS AND GLOSSARY OF TERMS

### Acronyms

| | |
|---|---|
| AC | Advisory Circular |
| AMJ | Advisory Material - Joint |
| CC1 | Control Category 1 |
| CC2 | Control Category 2 |
| COTS | Commercial off-the-Shelf |
| EUROCAE | European Organisation for Civil Aviation Equipment |
| FAA | Federal Aviation Administration |
| FAR | Federal Aviation Regulation |
| IC | Integrated Circuit |
| I/O | Input and/or Output |
| JAA | Joint Aviation Authorities |
| JAR | Joint Aviation Requirements |
| RTCA | RTCA, Inc. |
| SCI | Software Configuration Index |
| SCM | Software Configuration Management |
| SECI | Software Life Cycle Environment Configuration Index |
| SQA | Software Quality Assurance |

ANNEXE B

ACRONYMES ET GLOSSAIRE

Acronymes

| | |
|---|---|
| AC | Advisory Circular |
| AMJ | Advisory Material - Joint |
| CC1 | Control Category 1 |
| CC2 | Control Category 2 |
| COTS | Commercial off-the-Shelf |
| EUROCAE | European Organisation for Civil Aviation Equipment |
| FAA | Federal Aviation Administration |
| FAR | Federal Aviation Regulation |
| IC | Integrated Circuit |
| I/O | Input and/or Output |
| JAA | Joint Aviation Authorities |
| JAR | Joint Aviation Requirements |
| RTCA | RTCA, Inc. |
| SCI | Software Configuration Index |
| SCM | Software Configuration Management |
| SECI | Software Life Cycle Environment Configuration Index |
| SQA | Software Quality Assurance |

**Glossary of terms**

These definitions are provided for the terms which are used in this document. If a term is not defined in this annex, it is possible that it is defined instead in the body of this document. Refer to the American Heritage Dictionary for the definition of common terms.

Algorithm

A finite set of well-defined rules that gives a sequence of operations for performing a specific task

Anomalous behavior

Behavior that is inconsistent with specified requirements.

Applicant

A person or organization seeking approval from the certification authority.

Approval

The act or instance of expressing a favorable opinion or giving formal or official sanction.

Assurance

The planned and systematic actions necessary to provide adequate confidence and evidence that a product or process satisfies given requirements.

Audit

An independent examination of the software life cycle processes and their outputs to confirm required attributes.

Baseline

The approved, recorded configuration of one or more configuration items, that thereafter serves as the basis for further development, and that is changed only through change control procedures.

Certification

Legal recognition by the certification authority that a product, service, organization or person complies with the requirements. Such certification comprises the activity of technically checking the product, service, organization or person and the formal recognition of compliance with the applicable requirements by issue of a certificate, license, approval or other documents as required by national laws and procedures. In particular, certification of a product involves: (a) the process of assessing the design of a product to ensure that it complies with a set of standards applicable to that type of product so as to demonstrate an acceptable level of safety; (b) the process of assessing an individual product to ensure that it conforms with the certified type design; (c) the issuance of a certificate required by national laws to declare that compliance or conformity has been found with standards in accordance with items (a) or (b) above.

Certification Authority

The organization or person responsible within the state or country concerned with the certification of compliance with the requirements.

*NOTE: A matter concerned with aircraft, engine or propeller type certification or with equipment approval would usually be addressed by the certification authority; matters concerned with continuing airworthiness might be addressed by what would be referred to as the airworthiness authority.*

Certification credit

Acceptance by the certification authority that a process, product or demonstration satisfies a certification requirement.

**Glossaire**

Algorithme (Algorithm)

Ensemble fini de règles bien précises qui décrivent une séquence d'opérations nécessaire à l'exécution d'une tâche spécifique.

Analyse de couverture (Coverage analysis)

L'évaluation du degré auquel une activité du processus de vérification du logiciel satisfait ses objectifs.

Analyse de sécurité du système (System safety assessment)

Evaluation continue, systématique, et complète du système proposé dans le but de démontrer que les exigences liées à la sécurité sont satisfaites.

Analyseur statique (Static analyzer)

Outil logiciel aidant à mettre en évidence certaines propriétés d'un logiciel sans l'exécuter.

Approbation (Approval)

L'acte consistant à exprimer une opinion favorable ou à donner une autorisation formelle ou officielle.

Architecture du logiciel (Software architecture)

La structure du logiciel retenue pour mettre en oeuvre les spécifications du logiciel.

Architecture du système (System architecture)

La structure du matériel et du logiciel retenue pour mettre en oeuvre les spécifications du système.

Assurance (Assurance)

Actions planifiées et systématiques nécessaires pour fournir un niveau adéquat de confiance et de preuves qu'un produit ou un processus satisfait des exigences données.

Audit (Audit)

Examen indépendant des processus du cycle de vie du logiciel et de leurs sorties dans le but de confirmer les caractéristiques requises.

Autorité de certification (Certification Authority)

L'organisation ou la personne responsable, dans l'état ou le pays concerné, de la certification de la conformité aux exigences.

*NOTA :* *Les questions relatives à la certification de type d'un aéronef, d'un moteur, ou d'une hélice, ou à l'approbation d'un équipement seront en général traitées par l'Autorité de certification ; les questions relatives au maintien de la navigabilité pourront être traitées par ce qui sera considéré comme l'Autorité de navigabilité.*

Base de données (Database)

Ensemble de données, formant tout ou partie d'un autre ensemble de données, constitué par au moins un fichier, et suffisant pour un objectif donné ou pour un système de traitement de l'information déterminé.

Bibliothèque du logiciel (Software library)

Recueil organisé contenant les composants logiciels ainsi que les données et documents associés, conçu pour aider au développement, à l'utilisation, et aux modifications du logiciel. A titre d'exemple on peut citer les bibliothèques de développement, les bibliothèques de référence, les bibliothèques de production, les bibliothèques de programme et les bibliothèques d'archivage.

Change control

(1) The process of recording, evaluating, approving or disapproving and coordinating changes to configuration items after formal establishment of their configuration identification or to baselines after their establishment. (2) The systematic evaluation, coordination, approval or disapproval and implementation of approved changes in the configuration of a configuration item after formal establishment of its configuration identification or to baselines after their establishment.

*NOTE: This term may be called configuration control in other industry standards.*

Code

The implementation of particular data or a particular computer program in a symbolic form, such as source code, object code or machine code.

Commercial off-the-shelf (COTS) software

Commercially available applications sold by vendors through public catalog listings. COTS software is not intended to be customized or enhanced. Contract-negotiated software developed for a specific application is not COTS software.

Compiler

Program that translates source code statements of a high level language, such as FORTRAN or Pascal, into object code.

Component

A self-contained part, combination of parts, sub-assemblies or units, which performs a distinct function of a system.

Condition

A Boolean expression containing no Boolean operators.

Condition/Decision Coverage

Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken on all possible outcomes at least once, and every decision in the program has taken on all possible outcomes at least once.

Configuration identification

(1) The process of designating the configuration items in a system and recording their characteristics. (2) The approved documentation that defines a configuration item.

Configuration item

(1) One or more hardware or software components treated as a unit for configuration management purposes. (2) Software life cycle data treated as a unit for configuration management purposes.

Configuration management

(1) The process of identifying and defining the configuration items of a system, controlling the release and change of these items throughout the software life cycle, recording and reporting the status of configuration items and change requests and verifying the completeness and correctness of configuration items. (2) A discipline applying technical and administrative direction and surveillance to (a) identify and record the functional and physical characteristics of a configuration item, (b) control changes to those characteristics, and (c) record and report change control processing and implementation status.

Configuration status accounting

The recording and reporting of the information necessary to manage a configuration effectively, including a listing of the approved configuration identification, the status of proposed changes to the configuration and the implementation status of approved changes.

Certification (Certification)

Reconnaissance légale, par l'Autorité de certification, du fait qu'un produit, un service, une organisation, ou une personne est conforme aux exigences. Une telle certification comprend l'activité de contrôle technique du produit, du service, de l'organisation, ou de la personne et la reconnaissance formelle de la conformité avec les exigences applicables par émission d'un certificat, d'une autorisation, d'une approbation, ou de tout autre document en fonction des lois et procédures nationales. En particulier, la certification d'un produit implique : (a) le processus d'évaluation de la conception d'un produit dans le but de garantir qu'il est conforme à un ensemble de règles applicables à ce type de produit afin de démontrer un niveau acceptable de sécurité ; (b) le processus d'évaluation d'un produit individuel dans le but de garantir qu'il est conforme à la définition de type homologuée ; (c) l'émission d'un certificat exigé par les lois nationales, qui déclare qu'on a constaté la conformité avec les normes en accord avec les points (a) et (b) ci-dessus.

Classe d'équivalence (Equivalence class)

La partition du domaine d'entrée d'un programme, de telle sorte qu'un test utilisant une valeur représentative d'une classe est équivalent à un test utilisant une autre valeur de cette classe.

Code (Code)

La représentation sous forme symbolique des données ou d'un programme de calculateur particuliers, tels que code source, code objet, ou code machine.

Code désactivé (Deactivated code)

Code objet exécutable (ou données) qui, par conception, soit (a) n'est pas prévu pour être exécuté (ou utilisées), par exemple une partie d'un composant logiciel développé antérieurement, soit (b) n'est exécuté (ou utilisées) que dans certaines configurations de l'environnement de la machine cible, par exemple un code activable selon la valeur d'une broche matérielle ou en fonction d'options programmées par logiciel.

Code mort (Dead code)

Code objet exécutable (ou données) qui, en raison d'une erreur de conception, ne peut être exécuté (ou utilisées) dans une configuration opérationnelle de l'environnement de la machine cible et qui n'est pas traçable vers une spécification du système ou du logiciel. Ne sont pas considérés les identificateurs incorporés au logiciel.

Code objet (Object code)

Représentation de bas niveau du programme informatique, qui n'est généralement pas directement exécutable par la machine cible, mais qui comprend les informations d'adressage relatif en plus des instructions d'exécution du programme par le processeur.

Code source (Source code)

Code écrit dans un langage source, tels que langage assembleur ou langage de haut niveau, acceptable par un assembleur ou un compilateur.

Compilateur (Compiler)

Programme qui traduit en code objet les instructions d'un code source écrit en langage de haut niveau, tel que FORTRAN ou Pascal.

Composant (Component)

Partie indivisible, combinaison de parties, sous-ensembles ou éléments unitaires, qui réalisent une fonction distincte d'un système.

Condition (Condition)

Expression booléenne ne contenant pas d'opérateurs booléens.

Condition de panne (Failure Condition)

L'effet sur l'aéronef et ses occupants, à la fois direct et indirect, causé par ou auquel contribuent une ou plusieurs pannes, en prenant en considération les conditions défavorables de fonctionnement et d'environnement applicables. Une condition de panne se classe en fonction de l'importance de son effet comme défini dans les documents FAA AC 25.1309-1A ou JAA AMJ 25.1309.

Control coupling

The manner or degree by which one software component influences the execution of another software component.

Control program

A computer program designed to schedule and to supervise the execution of programs in a computer system.

Coverage analysis

The process of determining the degree to which a proposed software verification process activity satisfies its objective.

Database

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.

Data coupling

The dependence of a software component on data not exclusively under the control of that software component.

Data dictionary

The detailed description of data, parameters, variables, and constants used by the system.

Data type

A class of data, characterized by the members of the class and the operations that can be applied to them. Examples are character types and enumeration types.

Deactivated code

Executable object code (or data) which by design is either (a) not intended to be executed (code) or used (data), for example, a part of a previously developed software component, or (b) is only executed (code) or used (data) in certain configurations of the target computer environment, for example, code that is enabled by a hardware pin selection or software programmed options.

Dead code

Executable object code (or data) which, as a result of a design error cannot be executed (code) or used (data) in a operational configuration of the target computer environment and is not traceable to a system or software requirement. An exception is embedded identifiers.

Decision

A Boolean expression composed of conditions and zero or more Boolean operators. A decision without a Boolean operator is a condition. If a condition appears more than once in a decision, each occurrence is a distinct condition.

Decision Coverage

Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once.

Derived requirements

Additional requirements resulting from the software development processes, which may not be directly traceable to higher level requirements.

Emulator

A device, computer program, or system that accepts the same inputs and produces the same output as a given system using the same object code.

Equivalence class

The partition of the input domain of a program such that a test of a representative value of the class is equivalent to a test of other values of the class.

Couplage par les flux de contrôle (Control coupling)

La manière ou le degré avec lequel un composant logiciel influence le fonctionnement d'un autre composant logiciel.

Couplage par les flux de données (Data coupling)

La dépendance d'un composant logiciel vis à vis de données qui ne sont pas exclusivement maîtrisées par ce composant.

Couverture des conditions/décisions (Condition/Decision Coverage)

Chaque point d'entrée et de sortie du programme a été mis en oeuvre au moins une fois, chaque condition d'un point de décision du programme a produit au moins une fois tous les résultats possibles, et chaque décision du programme a produit au moins une fois tous les résultats possibles.

Couverture des décisions (Decision coverage)

Chaque point d'entrée et de sortie du programme a été mis en oeuvre au moins une fois et chaque décision du programme a produit au moins une fois tous les résultats possibles.

Couverture des instructions (Statement coverage)

Chaque instruction du programme a été mise en oeuvre au moins une fois.

Couverture modifiée de condition/décision (Modified condition/decision coverage)

Chaque point d'entrée et de sortie du programme a été mis en oeuvre au moins une fois, chaque condition d'un point de décision du programme a produit au moins une fois tous les résultats possibles, chaque décision du programme a produit au moins une fois tous les résultats possibles, et on a démontré que chaque condition dans une décision affecte de manière indépendante le résultat de cette décision. On démontre qu'une condition affecte de manière indépendante le résultat d'une décision en ne faisant varier que cette condition et en laissant inchangées toutes les autres conditions.

Crédit de certification (Certification Credit)

Reconnaissance par l'Autorité de certification qu'un processus, un produit, ou une démonstration satisfait une exigence de certification.

Critères de transition (Transition criteria)

Les conditions minimales à satisfaire, définies lors de la planification du logiciel, pour engager un processus.

Cycle de vie du logiciel (Software life cycle)

(1) Ensemble ordonné des processus déterminés par une organisation comme étant suffisants et adéquats pour réaliser un produit logiciel. (2) Période de temps qui commence avec la décision de réaliser ou de modifier un produit logiciel et qui s'achève quand le produit est retiré du service.

Décision (Decision)

Expression booléenne composée de conditions et éventuellement d'opérateurs booléens. Une décision sans opérateurs booléens est une condition. Si une condition apparaît plus d'une fois dans une décision, chaque occurrence est une condition distincte.

Défaut (Fault)

Manifestation d'une erreur dans le logiciel. Un défaut, en se produisant, peut entraîner une panne.

Dictionnaire des données (Data dictionary)

La description détaillée des données, paramètres, variables, et constantes utilisés par le système.

Dysfonctionnement (Anomalous behavior)

Comportement non cohérent avec les exigences spécifiées.

Elément de Configuration (Configuration item)

(1) Un ou plusieurs composants du matériel ou du logiciel, considérés comme une unité du point de vue de la gestion de configuration. (2) Données du cycle de vie du logiciel considérées comme une unité du point de vue de la gestion de configuration.

Error

With respect to software, a mistake in requirements, design or code.

Failure

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

Failure condition

The effect on the aircraft and its occupants both direct and consequential, caused or contributed to by one or more failures, considering relevant adverse operational and environmental conditions. A failure condition is classified according to the severity of its effect as defined in FAA AC 25.1309-1A or JAA AMJ 25.1309.

Fault

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

Fault tolerance

The built-in capability of a system to provide continued correct execution in the presence of a limited number of hardware or software faults.

Formal methods

Descriptive notations and analytical methods used to construct, develop and reason about mathematical models of system behavior.

Hardware/software integration

The process of combining the software into the target computer.

High-level requirements

Software requirements developed from analysis of system requirements, safety-related requirements, and system architecture.

Host computer

The computer on which the software is developed.

Independence

Separation of responsibilities which ensures the accomplishment of objective evaluation. (1) For software verification process activities, independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified, and a tool(s) may be used to achieve an equivalence to the human verification activity. (2) For the software quality assurance process, independence also includes the authority to ensure corrective action.

Integral process

A process which assists the software development processes and other integral processes and, therefore, remains active throughout the software life cycle. The integral processes are the software verification process, the software quality assurance process, the software configuration management process, and the certification liaison process.

Interrupt

A suspension of a task, such as the execution of a computer program, caused by an event external to that task, and performed in such a way that the task can be resumed.

Low-level requirements

Software requirements derived from high-level requirements, derived requirements, and design constraints from which source code can be directly implemented without further information.

Emulateur (Emulator)

Dispositif, programme informatique ou système acceptant les mêmes données d'entrée et produisant les mêmes sorties qu'un système donné en utilisant le même code objet.

Erreur (Error)

En ce qui concerne les logiciels, erreur de spécification, de conception, ou de codage.

Exigences de bas niveau (Low-level requirements)

Spécifications du logiciel obtenues à partir des exigences de haut niveau, des exigences dérivées, et des contraintes de conception, à partir desquelles le code source peut être réalisé directement sans information supplémentaire.

Exigences de haut niveau (High-level requirements)

Spécifications du logiciel développées à partir de l'analyse des spécifications du système, des exigences liées à la sécurité, et de l'architecture du système.

Exigences dérivées (Derived requirements)

Exigences complémentaires résultant des processus de développement du logiciel, qui peuvent ne pas être directement traçables vers des exigences de plus haut niveau.

Gestion de configuration (Configuration management)

(1) Le processus servant à identifier et à définir les éléments de configuration d'un système, à gérer la mise à disposition et les modifications de ces éléments pendant tout le cycle de vie du logiciel, à enregistrer et rendre compte de l'état des éléments de configuration et des demandes de modifications, et à vérifier l'exhaustivité et l'exactitude des éléments de configuration. (2) Activité d'orientation et de surveillance à la fois technique et administrative ayant pour but de a) identifier et enregistrer les caractéristiques fonctionnelles et physiques d'un élément de configuration, b) gérer les modifications apportées à ces caractéristiques, et (c) enregistrer et rendre compte de la gestion des modifications et de leur état de réalisation.

Gestion de modifications (Change control)

(1) Le processus d'enregistrement, d'évaluation, d'approbation ou de désapprobation, et de coordination des modifications apportées aux éléments de configuration d'un système postérieurement à leur identification formelle en configuration ou aux référentiels postérieurement à leur établissement. (2) L'évaluation, la coordination, l'approbation ou désapprobation, et la réalisation systématiques des modifications approuvées d'un élément de configuration postérieurement à son identification formelle en configuration ou des référentiels postérieurement à leur établissement.

*NOTA : Ce terme est parfois appelé contrôle de configuration dans d'autres normes industrielles.*

Historique du produit en service (Product service history)

Période continue pendant laquelle le logiciel a fonctionné dans un environnement connu, et les pannes successives ont été enregistrées.

Identification de configuration (Configuration identification)

(1) Le processus permettant de désigner des éléments de configuration d'un système et d'enregistrer leurs caractéristiques. (2) La documentation approuvée qui définit un élément de configuration.

Indépendance (Independence)

Séparation des responsabilités qui garantit l'exécution d'une évaluation objective. (1) Pour les activités du processus de vérification du logiciel, l'indépendance est obtenue quand l'activité de vérification est réalisée par une (ou des) personne(s) différente(s) du responsable du développement de l'élément à vérifier, et qu'on peut utiliser un (ou des) outil(s) pour réaliser une activité de vérification équivalente à celle de l'homme. (2) Pour le processus d'assurance qualité du logiciel, la notion d'indépendance suppose également que ceux qui sont en charge de ce processus soient investis de l'autorité garantissant que les actions correctives sont accomplies.

Intégration du logiciel (Software integration)

Le processus de regroupement des composants du code.

Means of compliance

The intended method(s) to be used by the applicant to satisfy the requirements stated in the certification basis for an aircraft or engine. Examples include statements, drawings, analyses, calculations, testing, simulation, inspection, and environmental qualification. Advisory material issued by the certification authority is used if appropriate.

Media

Devices or material which act as a means of transferal or storage of software, for example, programmable read-only memory, magnetic tapes or discs, and paper.

Memory device

An article of hardware capable of storing machine-readable computer programs and associated data. It may be an integrated circuit chip, a circuit card containing integrated circuit chips, a core memory, a disk, or a magnetic tape.

Modified Condition/Decision Coverage

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

Monitoring

(1) [Safety] Functionality within a system which is designed to detect anomalous behavior of that system. (2) [Quality Assurance] The act of witnessing or inspecting selected instances of test, inspection, or other activity, or records of those activities, to assure that the activity is under control and that the reported results are representative of the expected results. Monitoring is usually associated with activities done over an extended period of time where 100% witnessing is considered impractical or unnecessary. Monitoring permits authentication that the claimed activity was performed as planned.

Multiple-version dissimilar software

A set of two or more programs developed separately to satisfy the same functional requirements. Errors specific to one of the versions are detected by comparison of the multiple outputs.

Object Code

A low-level representation of the computer program not usually in a form directly usable by the target computer but in a form which includes relocation information in addition to the processor instruction information.

Part number

A set of numbers, letters or other characters used to identify a configuration item.

Patch

A modification to an object program, in which one or more of the planned steps of re-compiling, re-assembling or re-linking is bypassed. This does not include identifiers embedded in the software product, for example, part numbers and checksums.

Process

A collection of activities performed in the software life cycle to produce a definable output or product.

Product service history

A contiguous period of time during which the software is operated within a known environment, and during which successive failures are recorded.

Proof of correctness

A logically sound argument that a program satisfies its requirements.

Intégration matériel/logiciel (Hardware/software integration)

Le processus destiné à porter le logiciel dans la machine cible.

Interruption (Interrupt)

Suspension d'une tâche, telle que l'exécution d'un programme informatique, causée par un événement extérieur à cette tâche, et réalisée de manière à en permettre la reprise.

Jeu de test (Test case)

Ensemble des entrées de test, des conditions d'exécution, et des résultats attendus, développés pour un objectif particulier, tel que l'exécution d'un chemin particulier d'un programme ou la vérification de la conformité à une exigence particulière.

Logiciel (Software)

Programme informatique et, éventuellement, documentation et données associées concernant le fonctionnement d'un système informatique.

Logiciel à versions multiples dissimilaires (Multiple-version dissimilar software)

Ensemble de deux ou de plus de deux programmes développés séparément pour satisfaire les mêmes exigences fonctionnelles. Les erreurs spécifiques à une des versions sont détectées par comparaison des sorties de chaque version entre elles.

Logiciel du commerce sur étagère [COTS] (Commercial off-the shelf [COTS] software)

Applications disponibles dans le commerce sur catalogue. Les logiciels COTS ne sont pas prévus pour être personnalisés ou améliorés. Un logiciel négocié par contrat pour une application spécifique n'est pas un logiciel COTS.

Machine hôte (Host computer)

La machine sur laquelle le logiciel est développé.

Mémoire (Memory device)

Elément de matériel capable de stocker des programmes informatiques et les données associées exploitables par une machine. Ce peut être un circuit intégré, une carte contenant des circuits intégrés, une mémoire centrale, un disque, ou une bande magnétique.

Méthodes formelles (Formal methods)

Conventions de description et méthodes analytiques utilisées pour raisonner sur, construire, et développer les modèles mathématiques comportementaux d'un système.

Mise à disposition (Release)

L'action formelle de rendre disponible et d'autoriser l'utilisation d'un élément de configuration qu'il est possible de restaurer.

Modification de logiciel (Software change) :

Modification du code source, du code objet, du code objet exécutable, ou de la documentation associée, à partir du référentiel du logiciel.

Moyens de conformité (Means of compliance)

La ou les méthodes dont l'utilisation est prévue par le postulant pour satisfaire les exigences énoncées dans les bases de certification d'un aéronef ou d'un moteur. Les exemples comprennent les déclarations de conformité, les plans, les analyses, les calculs, les essais, la simulation, l'inspection, et la qualification d'environnement. Les circulaires d'application AC et AMJ émises par l'Autorité de certification sont utilisées le cas échéant.

Numéro de référence (Part number)

Ensemble de chiffres, lettres, ou autres caractères utilisés pour identifier un élément de configuration.

Outil logiciel (Software tool)

Programme informatique utilisé pour faciliter le développement, le test, l'analyse, la réalisation, ou la modification d'un autre programme ou de sa documentation, par exemple un outil de conception assistée, un compilateur, des outils de test, et des outils de modification.

Release

The act of formally making available and authorizing the use of a retrievable configuration item.

Reverse engineering

The method of extracting software design information from the source code.

Robustness

The extent to which software can continue to operate correctly despite invalid inputs.

Simulator

A device, computer program or system used during software verification, that accepts the same inputs and produces the same output as a given system, using object code which is derived from the original object code.

Software

Computer programs and, possibly, associated documentation and data pertaining to the operation of a computer system.

Software architecture

The structure of the software selected to implement the software requirements.

Software change

A modification in source code, object code, executable object code, or its related documentation from its baseline.

Software integration

The process of combining code components.

Software library

A controlled repository containing a collection of software and related data and documents designed to aid in software development, use or modification. Examples include software development library, master library, production library, program library and software repository.

Software life cycle

(1) An ordered collection of processes determined by an organization to be sufficient and adequate to produce a software product. (2) The period of time that begins with the decision to produce or modify a software product and ends when the product is retired from service.

Software partitioning

The process of separating, usually with the express purpose of isolating one or more attributes of the software, to prevent specific interactions and cross-coupling interference.

Software product

The set of computer programs, and associated documentation and data, designated for delivery to a user. In the context of this document, this term refers to software intended for use in airborne applications and the associated software life cycle data.

Software requirement

A description of what is to be produced by the software given the inputs and constraints. Software requirements include both high-level requirements and low-level requirements.

Software tool

A computer program used to help develop, test, analyze, produce or modify another program or its documentation. Examples are an automated design tool, a compiler, test tools and modification tools.

Source code

Code written in source languages, such as assembly language and/or high level language, in a machine-readable form for input to an assembler or a compiler.

Panne (Failure)

L'incapacité d'un système ou d'un composant d'un système à exécuter une fonction requise dans les limites spécifiées. Une panne peut survenir quand un défaut se produit.

Partitionnement du logiciel (Software partitioning)

La technique de ségrégation, utilisée généralement pour isoler des logiciels de caractéristiques différentes afin d'éviter les interactions spécifiques et les effets de bord parasites.

Patche (Patch)

Modification apportée à un programme objet, sans que soient reconduites toutes les étapes planifiées de compilation, d'assemblage, et d'édition de lien. Ceci ne comprend pas les identificateurs intégrés dans le produit logiciel, par exemple les numéros de référence et les sommes de contrôle.

Postulant (Applicant)

Personne ou organisation qui demande une approbation à l'Autorité de certification.

Preuve d'exactitude (Proof of correctness)

Argument logique et fondé selon lequel un programme satisfait ses exigences.

Procédure de test (Test procedure)

Instructions détaillées pour la mise en oeuvre et l'exécution d'un ensemble donné de jeux de test, et pour l'évaluation des résultats de leur exécution.

Processus (Process)

Ensemble d'activités réalisées au cours du cycle de vie du logiciel pour générer une sortie ou un produit défini.

Processus d'analyse de sécurité du système (System safety assessment process)

Les activités qui démontrent la conformité aux exigences de navigabilité et aux circulaires d'application associées, telles que JAA AMJ/FAA AC 25.1309. Les activités principales de ce processus comprennent en particulier l'évaluation fonctionnelle des risques, l'évaluation préliminaire de sécurité du système, et l'analyse de sécurité du système. Le degré de rigueur de ces activités dépend de la criticité, de la complexité, de la nouveauté, et de l'expérience en service appropriée du système concerné.

Processus intégral (Integral process)

Processus qui assiste les processus de développement du logiciel et les autres processus intégraux et, de ce fait, reste actif pendant tout le cycle de vie du logiciel. Les processus intégraux sont le processus de vérification du logiciel, le processus d'assurance qualité du logiciel, le processus de gestion de configuration du logiciel, et le processus de coordination pour la certification.

Produit logiciel (Software product)

L'ensemble de programmes informatiques, de leur documentation et des données associées, conçus pour être livrés à un utilisateur. Dans le contexte de ce document, ce terme se réfère aux logiciels conçus pour être utilisés dans les applications embarquées et aux données associées du cycle de vie du logiciel.

Programme de contrôle (Control program)

Programme informatique conçu pour ordonnancer et superviser l'exécution d'autres programmes dans un calculateur.

Qualification d'un outil (Tool qualification)

Le processus nécessaire à l'obtention d'un crédit de certification pour un outil logiciel dans le contexte d'un système embarqué spécifique.

Référentiel (Baseline)

La configuration approuvée et enregistrée d'un ou plusieurs éléments de configuration, servant ensuite de référence pour les développements ultérieurs, et qui ne peut être modifiée qu'au moyen de procédures de gestion des modifications.

Règles (Standard)

Directive ou base de comparaison utilisée à la fois pour fournir un guide de réalisation et le mode d'évaluation d'une activité ou du contenu d'une unité d'information particulière.

Standard

A rule or basis of comparison used to provide both guidance in and assessment of the performance of a given activity or the content of a specified data item.

Statement coverage

Every statement in the program has been invoked at least once.

Static analyzer

A software tool that helps to reveal certain properties of a program without executing the program.

Structure

A specified arrangement or interrelation of parts to form a whole.

System

A collection of hardware and software components organized to accomplish a specific function or set of functions.

System architecture

The structure of the hardware and the software selected to implement the system requirements.

System safety assessment

An ongoing, systematic, comprehensive evaluation of the proposed system to show that relevant safety-related requirements are satisfied.

System safety assessment process

Those activities which demonstrate compliance with airworthiness requirements and associated guidance material, such as, JAA AMJ/FAA AC 25.1309. The major activities within this process include: functional hazard assessment, preliminary system safety assessment, and system safety assessment. The rigor of the activities will depend on the criticality, complexity, novelty, and relevant service experience of the system concerned.

Task

The basic unit of work from the standpoint of a control program.

Test case

A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Testing

The process of exercising a system or system component to verify that it satisfies specified requirements and to detect errors.

Test procedure

Detailed instructions for the set-up and execution of a given set of test cases, and instructions for the evaluation of results of executing the test cases.

Tool qualification

The process necessary to obtain certification credit for a software tool within the context of a specific airborne system.

Traceability

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.

Transition criteria

The minimum conditions, as defined by the software planning process, to be satisfied to enter a process.

Rétroingéniérie (Reverse engineering)

La méthode consistant à extraire du code source des informations de conception ou de spécification du logiciel.

Robustesse (Robustness)

La mesure dans laquelle un logiciel peut continuer à fonctionner correctement malgré des données d'entrée invalides.

Simulateur (Simulator)

Dispositif, programme informatique ou système utilisé au cours de la vérification du logiciel, qui accepte les mêmes entrées et produit les mêmes sorties qu'un système donné, en utilisant un code objet dérivé du code objet d'origine.

Spécifications du logiciel (Software requirement)

Description du service fourni par le logiciel, compte tenu des entrées et des contraintes. Les spécifications du logiciel incluent les exigences de haut niveau et les exigences de bas niveau.

Structure (Structure)

Arrangement ou interconnexion particulier d'éléments formant un ensemble homogène.

Suivi des états de configuration (Configuration status accounting)

L'enregistrement et le compte rendu des informations nécessaires à la gestion effective de la configuration, comprenant une nomenclature de la configuration approuvée, l'état des propositions de modifications, et l'état de prise en compte des modifications approuvées.

Supports (Media)

Dispositifs ou matériau agissant comme moyen de transfert ou de stockage du logiciel, par exemple mémoires mortes programmables, bandes ou disques magnétiques, et papier.

Surveillance (Monitoring)

(1) [Sécurité] Fonctionnalité d'un système, conçue pour détecter tout dysfonctionnement de ce système. (2) [Assurance Qualité] Action d'attester ou d'inspecter des exemples choisis de tests, d'inspections, d'autres activités ou d'enregistrements de ces activités, afin de garantir que l'activité est contrôlée et que les résultats enregistrés sont représentatifs des résultats attendus. La surveillance est généralement associée à des activités exécutées sur une longue période de temps, où un contrôle à 100 % est considéré comme impossible à réaliser ou non nécessaire. Cette surveillance permet d'assurer que l'activité demandée a été exécutée comme prévu.

Système (System)

Ensemble de composants matériels et logiciels organisés de manière à exécuter une fonction ou un ensemble de fonctions spécifiques.

Tâche (Task)

L'unité élémentaire de travail du point de vue du programme de contrôle.

Test (Testing)

Le processus de mise à l'épreuve d'un système ou d'un composant de système dans le but de vérifier qu'il satisfait des exigences spécifiées et de détecter les erreurs.

Tolérance aux défauts (Fault tolerance)

La capacité intrinsèque d'un système à continuer à assurer un fonctionnement correct en présence d'un nombre limité de défauts de matériel ou de logiciel.

Traçabilité (Traceability)

La preuve d'une association entre éléments, telle qu'entre produits de processus, entre un produit et le processus dont il est issu, ou entre une exigence et sa mise en application.

Type de données (Data type)

Classe de données, incluant les éléments et les opérations qui leurs sont applicables. Les types caractère et les types énumérés en sont des exemples.

Validation

The process of determining that the requirements are the correct requirements and that they are complete. The system life cycle process may use software requirements and derived requirements in system validation.

Verification

The evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards provided to that process.

Validation (Validation)

Le processus assurant que les spécifications sont correctes et qu'elles sont complètes. Le processus du cycle de vie du système peut utiliser les spécifications du logiciel et les exigences dérivées pour la validation du système.

Vérification (Verification)

L'évaluation des produits d'un processus dans le but de garantir leur exactitude et leur cohérence vis à vis des données d'entrée et des règles appliquées à ce processus.

# APPENDIX A

## BACKGROUND OF DOCUMENT ED-12

### 1.0 PRIOR DOCUMENT VERSION HISTORY

In May 1980, the Radio Technical Commission for Aeronautics, now RTCA, Inc., established Special Committee 145 (SC-145), "Digital Avionics Software", to develop and document software practices that would support the development of software-based airborne systems and equipment. The European Organisation for Civil Aviation Electronics, now the European Organisation for Civil Aviation Equipment (EUROCAE), had previously established Working Group 12 (WG-12) to produce a similar document and, in October 1980, was ready to publish document ED-35, "Recommendations on Software Practice and Documentation for Airborne Systems." EUROCAE elected to withhold publication of its document and, instead, to work in concert with RTCA to develop a common set of guidelines. SC-145 produced RTCA Document DO-178, "Software Considerations in Airborne Systems and Equipment Certification", which was approved by the RTCA Executive Committee and published by RTCA in January 1982. EUROCAE published ED-12 shortly thereafter.

Early in 1983 the RTCA Executive Committee determined that DO-178 should be revised to reflect the experience gained in the certification of the aircraft and engines containing software-based systems and equipment. It established Special Committee 152 (SC-152) for this purpose.

As a result of this committee's work, a revised RTCA document, DO-178A, "Software Considerations in Airborne Systems and Equipment Certification", was published in 1985. Shortly thereafter, EUROCAE published ED-12A, which was identical in technical content to DO-178A.

### 2.0 RTCA / EUROCAE COMMITTEE ACTIVITIES IN THE PRODUCTION OF THIS DOCUMENT

In early 1989, the Federal Aviation Administration (FAA) formally requested that RTCA establish a Special Committee for the review and revision of DO-178A. Since its release in 1985, the aircraft manufacturers, the avionics industry and the certification authorities throughout the world have used DO-178A or the equivalent EUROCAE ED-12A as the primary source of the guidelines to determine the acceptability of systems and equipment containing software.

However, rapid advances in software technology, which were not envisioned by SC-152, and differing interpretations which were applied to some crucial areas, indicated that the guidelines required revision. Accordingly, an RTCA ad hoc group was formed with representatives from

ARINC, the Airline Pilots Association, the National Business Aircraft Association, the Air Transport Association and the FAA to consider the FAA request. The group reviewed the issues and experience associated with the application of DO-178A and concluded that a Special Committee should be authorized to revise DO-178A. The RTCA Executive Committee established Special Committee 167 (SC-167) during the autumn of 1989 to accomplish this task, and agreed to these Terms of Reference:

"Special Committee 167 shall review and revise, as necessary, RTCA Document DO-178A, "Software Considerations in Airborne Systems and Equipment Certification."

APPENDICE A

ORIGINES DU DOCUMENT EUROCAE ED-12B

## 1.0 HISTORIQUE DES VERSIONS ANTERIEURES DU DOCUMENT

En Mai 1980, la Commission Technique Radio pour l'Aéronautique, devenue maintenant la Société RTCA Inc., créait le Comité Spécial 145 (SC-145), "Logiciels pour l'avionique numérique", dans le but de développer et de rédiger des méthodes logicielles d'aide au développement de systèmes et équipements de bord utilisant du logiciel. L'Organisation Européenne pour l'Equipement Electronique de l'Aviation Civile, devenue maintenant Organisation Européenne pour l'Equipement de l'Aviation Civile (EUROCAE), avait créé auparavant le Groupe de Travail 12 (GT-12) dans le but de rédiger un document analogue et était prête, en Octobre 1980, à publier le document ED-35, "Recommandations sur les méthodes et la documentation des logiciels des systèmes embarqués". EUROCAE décida alors de suspendre la publication de son document et, en lieu et place, de travailler conjointement avec la RTCA au développement d'un ensemble commun de recommandations. Le SC-145 réalisait le Document RTCA DO-178, "Considérations Relatives aux Logiciels dans la Certification des Systèmes et Equipements de Bord", approuvé par le Comité Exécutif de la RTCA et publié par la RTCA en Janvier 1982. EUROCAE publiait peu après le Document ED-12.

Début 1983, le Comité Exécutif de la RTCA établissait la nécessité de réviser le DO-178 de manière à refléter l'expérience acquise dans la certification des aéronefs et des moteurs comportant des systèmes et des équipements à base de logiciel. Il créait alors le Comité Spécial 152 (SC-152) à cet effet.

Suite au travail de ce comité, un document RTCA révisé, le DO-178A, "Etude et Homologation du Logiciel des Systèmes et Equipements de Bord", était publié en 1985. Peu de temps après, EUROCAE publiait le document ED-12A dont le contenu technique était identique au DO-178A.

## 2.0 ACTIVITES DU COMITE RTCA/EUROCAE DANS L'ELABORATION DU PRESENT DOCUMENT

Au début de 1989, l'Administration Fédérale de l'Aviation (FAA) demandait formellement que la RTCA crée un Comité Spécial pour l'examen et la révision du DO-178A. Depuis sa publication en 1985, les constructeurs d'aéronefs, l'industrie de l'avionique, et les autorités de certification du monde entier avaient utilisé le DO-178A ou l'équivalent européen le ED-12A comme principale source des recommandations pour la détermination de l'acceptabilité des systèmes et équipements comportant du logiciel.

Cependant, les progrès rapides dans la technologie des logiciels, qui n'avaient pas été envisagés par le SC-152, et les différences d'interprétation qui avaient touché des domaines cruciaux, montraient qu'une révision des recommandations était indispensable. En conséquence de quoi, un groupe ad hoc RTCA était constitué avec des représentants de l'ARINC, de l'Association des Pilotes de Ligne, de l'Association Nationale des Entreprises d'Aéronautique, de l'Association du Transport Aérien, et de la FAA dans le but d'étudier la demande de la FAA. Le groupe examinait les problèmes et les expérience liées à l'utilisation du DO-178A et concluait qu'un Comité Spécial devait être autorisé à réviser le document. En automne 1989, le Comité Exécutif de la RTCA créait le Comité Spécial 167 (SC-167) pour réaliser cette tâche, et approuvait les attributions suivantes:

Le Comité Spécial 167 examinera et révisera, selon les besoins, le Document RTCA DO-178A, "Etude et Certification du Logiciel des Systèmes et Equipements de Bord".

GUIDANCE:

SC-167 should recognize the dynamic, evolving environment for software requirements, software design, code generation, testing and documentation; and formulate a revised document that can accommodate this environment while recommending suitably rigorous techniques. SC-167 should also recognize the international implications of this document and, therefore, should establish a close working relationship with EUROCAE, which has become the normal practice in RTCA committees. To accomplish this revision, the Special Committee should consider the experience gained through the field application of the guidance material contained in DO-178 and DO-178A, as well as the results of recent research in software engineering. The Special Committee should focus this review to address these areas:

1. Examine existing industry and government standards and consider for possible adaptation or reference, where relevant.
2. Assess the adequacy of existing software levels and the associated nature and degree of analysis, verification, test and assurance activities. The revised process criteria should be structured to support objective compliance demonstration.
3. Examine the criteria for tools to be used for certification credit, for example, tools for software development, software configuration management and software verification.
4. Examine the certification criteria for previously developed software, off-the-shelf software, databases and user-modifiable software for the system to be certified.
5. Examine the certification criteria for architectural and methodical strategies used to reduce the software level or to provide verification coverage, for example, partitioning and dissimilar software.
6. Examine configuration control guidelines, software quality assurance guidelines and identification conventions; and their compatibility with existing certification authority requirements for type certification, in-service modifications and equipment approval.
7. Consider the impact of new technology, such as, modular architecture, data loading, packaging, and memory technology.
8. Examine the need, content and delivery requirements of all documents, with special emphasis on the Software Accomplishment
9. Define and consider the interfaces between the software and system life cycles.
10. Review the criteria associated with making pre- and post-certification changes to a system.
11. Consider the impact of evolutionary development and other alternative life cycles to the model implied by DO-178A.

EUROCAE WG-12 was re-established to work with SC-167 and, to accomplish the task, five joint RTCA/EUROCAE working groups were formed to address these topics:

1. Documentation Integration and Production.
2. System Issues.
3. Software Development.
4. Software Verification.
5. Software Configuration Management and Software Quality Assurance.

The cooperative efforts of SC-167 and WG-12 culminated in the publication of RTCA document DO-178B and EUROCAE document ED-12B.

RECOMMANDATIONS :

Le SC-167 doit identifier l'environnement dynamique évolutif des spécifications du logiciel, de la conception du logiciel, de la génération de code, des tests, et de la documentation, et élaborer un document révisé, adapté à cet environnement tout en recommandant des techniques rigoureuses appropriées. Le SC-167 doit aussi identifier les implications internationales de ce document et, de ce fait, établir des relations de travail étroites avec EUROCAE, ce qui est devenu l'attitude normale des comités de la RTCA. Pour exécuter cette révision, le Comité Spécial doit prendre en considération l'expérience acquise grâce à l'utilisation pratique des recommandations figurant dans le DO-178 et le DO-178A, ainsi que les produits des recherches récentes en génie logiciel. Le Comité Spécial doit concentrer cet examen de manière à traiter les domaines suivants :

1. Examen des normes industrielles et gouvernementales existantes, et considération d'éventuelle adaptation ou prise en référence.
2. Evaluation de l'adéquation entre les niveaux logiciels existants et la nature et le degré associés des activités d'analyse, de vérification, d'essai, et d'assurance. Les critères de processus révisés doivent être structurés de manière à appuyer les démonstrations objectives de conformité.
3. Examen des critères concernant les outils utilisés pour les crédits de certification, par exemple les outils de développement du logiciel, de gestion de configuration du logiciel, et de vérification du logiciel.
4. Examen des critères de certification des logiciels développés antérieurement, des logiciels du commerce, des bases de données, et des logiciels modifiables par l'utilisateur pour le système à certifier.
5. Examen des critères de certification des stratégies d'architecture et de méthode utilisées pour réduire le niveau logiciel ou pour obtenir une couverture de vérification, par exemple le partitionnement et les logiciels dissimilaires.
6. Examen des recommandations de gestion de configuration, des recommandations d'assurance qualité, et des conventions d'identification; et de leur compatibilité avec les exigences existantes des autorités de certification pour les certifications de type, les modifications en service, et les approbations d'équipements.
7. Etude de l'impact des nouvelles technologies, telles qu'architecture modulaire, chargement de données, organisation interne, technologies des mémoires.
8. Examen de la nécessité, du contenu, et des exigences de distribution de tous les documents, en particulier du Résumé des Travaux Réalisés.
9. Définition et étude des interfaces entre le logiciel et les cycles de vie du système et du logiciel.
10. Examen des critères liés à l'exécution des modifications apportées à un système, antérieurement et postérieurement à la certification.
11. Etude de l'impact des développements évolutifs et d'autres cycles de vie sur le modèle considéré par DO-178A.

Le Groupe de Travail GT-12 d'EUROCAE était alors rétabli pour assurer un travail en commun avec le SC-167. Pour assurer cette tâche, cinq sous-groupes RTCA/EUROCAE étaient formés pour traiter les sujets suivants :

1. Intégration et réalisation de la documentation.
2. Questions relatives aux systèmes.
3. Développement du logiciel.
4. Vérification du logiciel.
5. Gestion de configuration du logiciel et assurance qualité.

Les efforts de coopération entre le SC-167 et le GT-12 ont abouti à la publication du document RTCA DO-178B/EUROCAE ED-12B.

## 3.0 SUMMARY OF DIFFERENCES BETWEEN ED-12B AND ED-12A

This is a complete rewrite of ED-12A.

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

ED-12B is primarily a process-oriented document. For each process, objectives are defined and a means of satisfying these objectives are described. A description of the software life cycle data which shows that the objectives have been satisfied is provided. The objectives for each process are summarized in tables and the effect of software level on the applicability and independence of these objectives is specified in the tables. The variation by software level in configuration management rigor for the software life cycle data is also in the tables.

ED-12B recognizes that many different software life cycles are acceptable for developing software for airborne systems and equipment. ED-12B emphasizes that the chosen software life cycle(s) should be defined during the planning for a project. The processes which comprise a software development project, no matter which software life cycle was chosen, are described. These processes fall into three categories: the software planning process, the software development processes, which include software requirements, software design, software coding and integration; and the integral processes which include software verification, software quality assurance, software configuration management and certification liaison. Integral processes are active throughout the software life cycle. ED-12B requires that criteria which control the transitions between software life cycle processes should be established during the software planning process.

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

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

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

A section covering additional topics has been added to provide guidance in areas previously not addressed in ED-12A. These topics include the use of previously developed software, tool qualification, and the use of alternative methods, including formal methods, exhaustive input testing, multiple-version dissimilar software verification, software reliability models, and product service history.

The glossary of terms has been reviewed for redundant or conflicting terminology and, where possible, industry-accepted definitions adopted.

The guidelines of this document were compared with international standards: ISO 9000-3 (1991), "Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software," and IEC 65A (Secretariat)122 (Draft - November 1991), "Software for Computers in the Application of Industrial Safety-Related Systems." These guidelines are considered to generally satisfy the intent of those standards.

A comprehensive index is provided to assist in the application of this document similar software versions detects actual errors or experience.

## 3.0 RESUME DES DIFFERENCES ENTRE LES DOCUMENTS ED-12B ET ED-12A

Il s'agit d'une réécriture complète de l'ED-12A.

Il est suggéré de lire le document en entier avant d'utiliser l'une de ses sections ou l'un des tableaux isolément.

L'ED-12B est avant tout un document orienté sur les processus. Pour chaque processus, les objectifs sont définis et les moyens de les satisfaire sont décrits. Une description des données du cycle de vie du logiciel est fournie, qui démontre que les objectifs ont été satisfaits. Les tableaux résument les objectifs de chaque processus et spécifient l'effet du niveau logiciel sur l'applicabilité et l'indépendance de ces objectifs. La variation par niveau logiciel de la rigueur de la gestion de configuration pour les données du cycle de vie du logiciel figure aussi dans les tableaux.

L'ED-12B reconnaît que de nombreux et différents cycles de vie du logiciel sont acceptables pour le développement des logiciels destinés aux systèmes et équipements de bord. L'ED-12B insiste sur le fait que le(s) cycle(s) de vie du logiciel choisi(s) doit(vent) être défini(s) au cours de la planification d'un projet. On décrit les processus qui constituent un projet de développement de logiciel, quel que soit le cycle de vie du logiciel choisi. Ces processus se classent en trois catégories: le processus de planification du logiciel, le processus de développement du logiciel qui comprend les exigences, la conception, le codage du logiciel, et l'intégration; et les processus intégraux qui comprennent la vérification, l'assurance qualité, et la gestion de configuration du logiciel, ainsi que la coordination pour la certification. Les processus intégraux sont actifs pendant tout le cycle de vie du logiciel. L'ED-12B exige que les critères qui régissent les transitions entre les processus du cycle de vie du logiciel soient établis au cours du processus de planification du logiciel.

La relation entre les processus du cycle de vie du logiciel et les processus du cycle de vie du système est définie plus complètement. Il est nécessaire de prendre en considération, au cours du processus d'analyse de sécurité du système, les conditions de panne associées aux fonctions devant être assurées par le logiciel. Ceci est la base de la détermination du niveau logiciel. Les niveaux logiciels sont maintenant le Niveau A, le Niveau B, le Niveau C, le Niveau D, et le Niveau E.

La section traitant de la vérification des logiciels insiste sur les essais fondés sur les exigences, avec en outre la couverture structurelle.

Les éléments à fournir et les autres unités d'information sont définis plus complètement, ainsi que la gestion de configuration requise.

Il a été ajouté une section traitant de sujets complémentaires dans le but de fournir des recommandations dans des domaines non encore traités dans l'ED-12A. Ces sujets comprennent l'emploi de logiciels développés antérieurement, la qualification d'outil, et l'usage d'autres méthodes, y compris les méthodes formelles, les tests exhaustifs de données d'entrée, la vérification des logiciels à versions multiples dissimilaires, les modèles de fiabilité de logiciels, et l'historique du produit en service.

Le glossaire des termes a été révisé en vue d'éviter les terminologies redondantes ou conflictuelles et, dans la mesure du possible, les définitions acceptées par l'Industrie ont été adoptées.

Les recommandations de ce document ont été comparées avec les normes internationales ISO 9000-3 (1991), "Recommandations pour l'Application d'ISO 9001 au Développement, à la Fourniture, et à la Maintenance des Logiciels", et CEI 65A (Secrétariat) 122 (Projet Novembre 1991), "Logiciels de Calculateurs pour l'Utilisation dans les Systèmes Industriels Liés à la Sécurité". Les présentes recommandations sont considérées comme satisfaisant en général le but recherché par ces normes.

APPENDIX B

COMMITTEE MEMBERSHIP

Chairmen

| | | |
|---|---|---|
| EUROCAE WG-12 | Daniel J. Hawkes | UK CAA |
| RTCA SC-167 | Roger A. Seeman | Boeing |

Secretaries

| | | |
|---|---|---|
| EUROCAE WG-12 | Jean-Michel Nogué | Aerospatiale |
| RTCA SC-167 | Michael DeWalt | FAA |

Working/Sub-Group Chairmen/Secretaries

Group 1

| | |
|---|---|
| B. Pflug | Boeing |

Group 2

| | |
|---|---|
| D. Allen | Boeing |
| M. De Walt | FAA |
| J-M. Nogué | Aerospatiale |

Group 3

| | |
|---|---|
| J. Krodel | Pratt & Whitney |
| N. Smith | British Aerospace |

Group 4

| | |
|---|---|
| J. Angermayer | Bendix/King ATAD |
| J-M. Astruc | Veridatas |
| G. Finelli | NASA |
| J. Williams | FAA |

Group 5

| | |
|---|---|
| T. Drake | British Aerospace |
| R. Hannan | Smiths Industries |
| M. Kress | Boeing |

Editorial Group

| | | | |
|---|---|---|---|
| A. Coupier | CEAT | B. Pflug | Boeing |
| R. Hannan | Smiths Industries | C. Secher | DGAC France |
| E. Kosowski | Rockwell Collins | N. Smith | British Aerospace |
| T. Kraft | FAA | J. Stesney | Litton Aero |
| F. Moyer | Rockwell Collins | W. Struck | Boeing |
| J-M. Nogué | Aerospatiale | L. Tripp | Boeing |
| K. Peterson | Honeywell | B. Weiser | Honeywell |

Federal Aviation Administrative Representative

| | |
|---|---|
| G. McIntyre | FAA Research and Development |

RTCA Representatives

J. Lohr
D. Watrous

EUROCAE Representative

B. Perret

APPENDICE B

COMITE MEMBRE

Chairmen

| | | |
|---|---|---|
| EUROCAE WG-12 | Daniel J. Hawkes | UK CAA |
| RTCA SC-167 | Roger A. Seeman | Boeing |

Secretaries

| | | |
|---|---|---|
| EUROCAE WG-12 | Jean-Michel Nogué | Aerospatiale |
| RTCA SC-167 | Michael DeWalt | FAA |

Working/Sub-Group Chairmen/Secretaries

Group 1

| | |
|---|---|
| B. Pflug | Boeing |

Group 2

| | |
|---|---|
| D. Allen | Boeing |
| M. De Walt | FAA |
| J-M. Nogué | Aerospatiale |

Group 3

| | |
|---|---|
| J. Krodel | Pratt & Whitney |
| N. Smith | British Aerospace |

Group 4

| | |
|---|---|
| J. Angermayer | Bendix/King ATAD |
| J-M. Astruc | Veridatas |
| G. Finelli | NASA |
| J. Williams | FAA |

Group 5

| | |
|---|---|
| T. Drake | British Aerospace |
| R. Hannan | Smiths Industries |
| M. Kress | Boeing |

Editorial Group

| | | | |
|---|---|---|---|
| A. Coupier | CEAT | B. Pflug | Boeing |
| R. Hannan | Smiths Industries | C. Secher | DGAC France |
| E. Kosowski | Rockwell Collins | N. Smith | British Aerospace |
| T. Kraft | FAA | J. Stesney | Litton Aero |
| F. Moyer | Rockwell Collins | W. Struck | Boeing |
| J-M. Nogué | Aerospatiale | L. Tripp | Boeing |
| K. Peterson | Honeywell | B. Weiser | Honeywell |

Federal Aviation Administrative Representative

| | |
|---|---|
| G. McIntyre | FAA Research and Development |

RTCA Representatives

J. Lohr
D. Watrous

EUROCAE Representative

B. Perret

MEMBERS

| | |
|---|---|
| Abell, K. | Hughes Information |
| Abraham, J. | University of Texas |
| Allocco, M. | AVIA TAC FAA TRW |
| Andrews, P. | Rockwell Collins |
| Angermayer, J. | Mitre Corporation |
| Ashpole, R. | British Aerospace Airbus |
| Atienza, L. | Innovative Solutions International Inc |
| van Baal, J. | RLD The Netherlands |
| Bailey, J. | Certification Services Inc |
| Bain, M. | Boeing |
| Baranes, S. | DGAC |
| Barrow, D. | Thomson-CSF |
| Bates, J. | CAA (SRG) ATS |
| Bauden, T. | FAA, AOS-240 |
| Bayram, N. | DASA KID –System |
| Bennett, P. | Virtual Prototypes Inc. |
| Beaufays, J. | Eurocontrol |
| Besnard, J. | Raytheon Systems Co. |
| Blakeley, A. | UK National Air Traffic Services Limited |
| Blankenberger, M. | Rolls-Royce Allison - Product Airworthiness |
| Bocvarov, S. | Innovative Solutions International, Inc. |
| Bosco, C. | FAA |
| Bradley, R. | Coleman Research Corporation |
| Bratteby-Ribbing, I. | FMV – Swedish Defence Materiel Administration |
| Brook, P. | Racal Avionics |
| Brookshire, J. | Lockheed Martin Aeronautical Systems |
| Brown, D. | Rolls-Royce PLC |
| Brown, P. | MOD (UK) |
| Bryant, J. | RTCA Inc |
| Bull, R. | ERA Technology |
| Bureau, J. | Magellan Systems Corp |
| Carlier, Y. | DGAC |
| Chang, G. | Air Economics Group Inc |
| Chelini, J. | Aonix |
| Clarke, G. | Honeywell |
| Coleman, J. | Hamilton Standard |
| Combes, N. | Fairchild/Dornier |
| Conrad, R. | Lockheed Martin |
| Cook, P. | Sigma Associates (Aerospace) |
| Cooke, D. | Penny and Giles |
| Cooley, R. | FAA |
| Cosimini, C. | European Space Agency |
| Cullimore, S. | Smiths Industries Aerospace |
| Dangu, M. | Eurologiciel |
| Danner, B. | TRW |
| Davies, P. | Logica Space Division |
| Delseny, H. | Aerospatiale |
| Depoy, N. | TRW/SETA |
| Desquilbet, C. | CEAT/EL |
| Devanbu, P. | University of California |
| Devlin, P. | Ultra Electronics |
| DeWalt, M. | Certification Services Inc. |
| Dolman, B. | Lucas Aerospace |
| Dorneau, N. | Sextant Avionique |
| Dorsey, C. | Digital Flight |
| Dunn, P. | Litton Aero Products |
| Durand, | Aerospatiale |
| Duthu, L. | Thomson-CSF |
| Eaton, A. | CAA (SRG) ATS |
| Egron, D. | AIRSYS ATM |
| Eichner, J-M. | Ashtech |
| Elliott, B | Harris ISD, TMS BAT |
| Escaffre, F. | Bureau Veritas - Veridatas |
| Evans, R. | Pratt & Whitney Canada |
| Eveleens, R. | National Aerospace Laboratory |
| Ferrell, T. | SAIC |
| Ferrell, U. | Mitre Corporation |
| Fiduccia, P. | Small Aircraft Manufacturers Association |
| Finelli, G. | NASA Langley Research Center |
| Fisher, D. | ANPC |
| Fisk, F. | GE Aircraft Engines |
| Fridrick, S. | GE Aircraft Engines |
| Fritts, J. | AlliedSignal Aerospace |
| Furlo, F. | Coleman Research Corporation |
| Furstnau, R. | Rolls-Royce Allison |
| Gagnaire, J-P. | SNECMA/ELECMA |
| Gasiorowski, M. | Honeywell BCAS |
| Germanicus, P. | Intertechnique |
| Gladstone, J. | Lockheed Martin Aero Systems |
| Gneiting, K. | IABG mbH |
| Goodchild, P. | Stewart Hughes Limited |
| Grimal, F. | EUROCAE |
| Grabowski, D. | General Digital |
| Gundlach, B. | BF Goodrich Avionic Systems |
| Hall, R. | Harris Corporation |
| Hamann, A. | FAA, ASW-170 |
| Hannan, R. | Sigma Associates (Aerospace) |
| Hansen, J. | Raytheon Aircraft Montek |
| Hatfield, D. | FAA, AOS-240 |
| Hawken, D. | UK National Air Traffic Services |
| Hayhurst, K. | NASA Langley Research Center |
| Haynes, G. | Rockwell-Collins (UK) |
| Heller, P. | DASA-Airbus |
| Henderson, G. | Ametek Aerospace |
| Hill, K. | Independent Consultant |
| Hokans, R. | Honeywell ATS |
| Holloway, C. M. | NASA Langley Research |

MEMBRES

| | |
|---|---|
| Abell, K. | Hughes Information |
| Abraham, J. | University of Texas |
| Allocco, M. | AVIA TAC FAA TRW |
| Andrews, P. | Rockwell Collins |
| Angermayer, J. | Mitre Corporation |
| Ashpole, R. | British Aerospace Airbus |
| Atienza, L. | Innovative Solutions International Inc |
| van Baal, J. | RLD The Netherlands |
| Bailey, J. | Certification Services Inc |
| Bain, M. | Boeing |
| Baranes, S. | DGAC |
| Barrow, D. | Thomson-CSF |
| Bates, J. | CAA (SRG) ATS |
| Bauden, T. | FAA, AOS-240 |
| Bayram, N. | DASA KID –System |
| Bennett, P. | Virtual Prototypes Inc. |
| Beaufays, J. | Eurocontrol |
| Besnard, J. | Raytheon Systems Co. |
| Blakeley, A. | UK National Air Traffic Services Limited |
| Blankenberger, M. | Rolls-Royce Allison - Product Airworthiness |
| Bocvarov, S. | Innovative Solutions International, Inc. |
| Bosco, C. | FAA |
| Bradley, R. | Coleman Research Corporation |
| Bratteby-Ribbing, I. | FMV – Swedish Defence Materiel Administration |
| Brook, P. | Racal Avionics |
| Brookshire, J. | Lockheed Martin Aeronautical Systems |
| Brown, D. | Rolls-Royce PLC |
| Brown, P. | MOD (UK) |
| Bryant, J. | RTCA Inc |
| Bull, R. | ERA Technology |
| Bureau, J. | Magellan Systems Corp |
| Carlier, Y. | DGAC |
| Chang, G. | Air Economics Group Inc |
| Chelini, J. | Aonix |
| Clarke, G. | Honeywell |
| Coleman, J. | Hamilton Standard |
| Combes, N. | Fairchild/Dornier |
| Conrad, R. | Lockheed Martin |
| Cook, P. | Sigma Associates (Aerospace) |
| Cooke, D. | Penny and Giles |
| Cooley, R. | FAA |
| Cosimini, C. | European Space Agency |
| Cullimore, S. | Smiths Industries Aerospace |
| Dangu, M. | Eurologiciel |
| Danner, B. | TRW |
| Davies, P. | Logica Space Division |
| Delseny, H. | Aerospatiale |
| Depoy, N. | TRW/SETA |
| Desquilbet, C. | CEAT/EL |
| Devanbu, P. | University of California |
| Devlin, P. | Ultra Electronics |
| DeWalt, M. | Certification Services Inc. |
| Dolman, B. | Lucas Aerospace |
| Dorneau, N. | Sextant Avionique |
| Dorsey, C. | Digital Flight |
| Dunn, P. | Litton Aero Products |
| Durand, | Aerospatiale |
| Duthu, L. | Thomson-CSF |
| Eaton, A. | CAA (SRG) ATS |
| Egron, D. | AIRSYS ATM |
| Eichner, J-M. | Ashtech |
| Elliott, B | Harris ISD, TMS BAT |
| Escaffre, F. | Bureau Veritas - Veridatas |
| Evans, R. | Pratt & Whitney Canada |
| Eveleens, R. | National Aerospace Laboratory |
| Ferrell, T. | SAIC |
| Ferrell, U. | Mitre Corporation |
| Fiduccia, P. | Small Aircraft Manufacturers Association |
| Finelli, G. | NASA Langley Research Center |
| Fisher, D. | ANPC |
| Fisk, F. | GE Aircraft Engines |
| Fridrick, S. | GE Aircraft Engines |
| Fritts, J. | AlliedSignal Aerospace |
| Furlo, F. | Coleman Research Corporation |
| Furstnau, R. | Rolls-Royce Allison |
| Gagnaire, J-P. | SNECMA/ELECMA |
| Gasiorowski, M. | Honeywell BCAS |
| Germanicus, P. | Intertechnique |
| Gladstone, J. | Lockheed Martin Aero Systems |
| Gneiting, K. | IABG mbH |
| Goodchild, P. | Stewart Hughes Limited |
| Grimal, F. | EUROCAE |
| Grabowski, D. | General Digital |
| Gundlach, B. | BF Goodrich Avionic Systems |
| Hall, R. | Harris Corporation |
| Hamann, A. | FAA, ASW-170 |
| Hannan, R. | Sigma Associates (Aerospace) |
| Hansen, J. | Raytheon Aircraft Montek |
| Hatfield, D. | FAA, AOS-240 |
| Hawken, D. | UK National Air Traffic Services |
| Hayhurst, K. | NASA Langley Research Center |
| Haynes, G. | Rockwell-Collins (UK) |
| Heller, P. | DASA-Airbus |
| Henderson, G. | Ametek Aerospace |
| Hill, K. | Independent Consultant |
| Hokans, R. | Honeywell ATS |
| Holloway, C. M. | NASA Langley Research |

| | |
|---|---|
| | Center |
| Holmer, P. | Systems Resources Corp. |
| Hutchinson, P. | GEC Marconi |
| Jackson, R. | Raytheon Systems Company |
| Jacobs, J. | AVROTEC |
| Jaffe, M. | Embry-Riddle Aeronautical University |
| Jaglarz , M. | Canadair |
| Janelle, J. | Honeywell |
| Joag, V. | Smiths Industries |
| Johnson, L. | Boeing Commercial Airplane Group |
| Jones, C. | Bombardier Aerospace - deHavilland Division |
| Jones, R. | FAA Atlanta ACO |
| Joseph, M. | Mitre Corporation (CAASD) |
| Kantkaz, R. | Oracle Corporation |
| Kanuritch, N. | BAe Systems and Services |
| Karoun, O. | SNECMA-ELECMA |
| Kerr, J. | Smiths Industries |
| Key, R. | FAA/AOS-240 |
| Kinney, D. | Boeing Company |
| Kleine-Beek, W. | Luftfahrt-Bundesamt (LBA) |
| Kosowski, E. | Rockwell Collins Avionics |
| Kraft, T. | Federal Aviation Administration |
| Kress, M. | Boeing Company |
| Krodel, J. | United Technologies Research Center |
| Kruth, D. | Oracle Corporation |
| Lachambre, J-P. | Aerospatiale APSYS |
| Ladier, G. | Aerospatiale |
| Lambalot, R. | Boeing |
| Lampton, M. | Rockwell-Collins |
| Lee, A. | Advanced Navigation and Positioning Corporation (ANPC) |
| Le Gac, P. | Bombardier Aerospace - Canadair |
| Lebacqz, V. | NASA |
| Leung, H. | Canadian Marconi Company |
| Lillis, M. | Hamilton Standard –UTC |
| Liu, M. | Mitre Corporation |
| Lomax, S. | Magellan Systems Corporation |
| Lowe, H. | Smiths Industries Aerospace |
| Lynch, J. | Honeywell |
| Maclellan, A. | Honeywell |
| Marcoux, J. | European Space Agency |
| Martinez, L. | Boeing |
| Masters, D. | Pratt & Whitney Aircraft |
| Matheson, W. | Camber Corp |
| Matthews, P. | Pratt and Whitney Canada Inc |
| Matthews, S. | Avidyne Corp |
| McAnany, I. | Dornier Satellite Systems |
| McCallister, R. | FAA |
| McCormick, F. | Certification Services Inc. |
| McGookey, J. | Smiths Industries |
| McIntyre, G. | FAA, ASY300 |
| McKenna, P. | Smiths Industries Aerospace |
| McKinlay, A. | Booz Allen & Hamilton Inc. |
| McLoughlin, F. | Seagull Technology |
| Meer, M. | Compaq/Digital Computer Corp |
| Melet, V. | DGAC/STNA 3ED |
| Mendez, R. | FAA |
| Meyers, J. | FAA |
| Miles, C. | FAA |
| Moore, P. | Civil Aviation Authority |
| Morgan, S. | MoD (PE) |
| Mortensen, U. | European Space Agency |
| Moyer, F. | Rockwell Collins Avionics |
| Mueller, W. | Bodenseewerk Geratetechnik GmbH (BGT) |
| Muench, M. | Trimble Navigation |
| Mugridge, C. | Defence Evaluation Research Agency |
| Murthi, A. | Strategic Technology Institute |
| Nash, D. | Raytheon Systems |
| Neill, W. | Harris Corporation |
| Nickerson, S. | GE Aircraft Engines |
| Norris, J. | Eldec Corporation |
| Obeid, S. | AlliedSignal Aerospace |
| O'Neil, K. | Advanced Aviation Technology |
| Nogue, J-M. | Aerospatiale |
| Osborg, G. | Normarc AS |
| Ostrom, G. | Allied Signal CAS |
| Paasch, S. | Federal Aviation Administration |
| Paly, G. | SNECMA-ELECMA |
| Pehrson, R. | Boeing |
| Pierce, R. | York Software Engineering |
| Penny, J. | Independent Consultant |
| Perret, B. | EUROCAE |
| Petersen, J. | FAA |
| Pflug, B. | Boeing |
| Phan, T. | FAA, ANM-130L |
| Pilj, G. | Learjet |
| Piper, E. | Boeing - Long Beach Division |
| Porzi, M. | ELMER S.p.A. |
| Pothon, F. | Turbomeca |
| Prisaznuk, P. | AEEC ARINC |
| Pro, B. | Allied Signal EAS |
| Pumfrey, J. | CAA SRG ATS |
| Pyster, A. | FAA |
| Ramsey, M. | Allied Signal EAS |
| Rao, R. | Transport Canada |
| Rebai, K. | |
| Reeve, T. | AVTECH Corp. |
| Reich, A. | Racal Avionics |
| Rhoads, T. | BF Goodrich Aerospace |
| Richardson, M. | Cessna Aircraft Co. |
| Rierson, L. | FAA |
| Rigby, K. | Independent Consultant |
| Rodrigues, S. | Thomson Simulation and Training |
| Rogness, L. | Rockwell Collins |

| | |
|---|---|
| | Center |
| Holmer, P. | Systems Resources Corp. |
| Hutchinson, P. | GEC Marconi |
| Jackson, R. | Raytheon Systems Company |
| Jacobs, J. | AVROTEC |
| Jaffe, M. | Embry-Riddle Aeronautical University |
| Jaglarz , M. | Canadair |
| Janelle, J. | Honeywell |
| Joag, V. | Smiths Industries |
| Johnson, L. | Boeing Commercial Airplane Group |
| Jones, C. | Bombardier Aerospace - deHavilland Division |
| Jones, R. | FAA Atlanta ACO |
| Joseph, M. | Mitre Corporation (CAASD) |
| Kantkaz, R. | Oracle Corporation |
| Kanuritch, N. | BAe Systems and Services |
| Karoun, O. | SNECMA-ELECMA |
| Kerr, J. | Smiths Industries |
| Key, R. | FAA/AOS-240 |
| Kinney, D. | Boeing Company |
| Kleine-Beek, W. | Luftfahrt-Bundesamt (LBA) |
| Kosowski, E. | Rockwell Collins Avionics |
| Kraft, T. | Federal Aviation Administration |
| Kress, M. | Boeing Company |
| Krodel, J. | United Technologies Research Center |
| Kruth, D. | Oracle Corporation |
| Lachambre, J-P. | Aerospatiale APSYS |
| Ladier, G. | Aerospatiale |
| Lambalot, R. | Boeing |
| Lampton, M. | Rockwell-Collins |
| Lee, A. | Advanced Navigation and Positioning Corporation (ANPC) |
| Le Gac, P. | Bombardier Aerospace - Canadair |
| Lebacqz, V. | NASA |
| Leung, H. | Canadian Marconi Company |
| Lillis, M. | Hamilton Standard –UTC |
| Liu, M. | Mitre Corporation |
| Lomax, S. | Magellan Systems Corporation |
| Lowe, H. | Smiths Industries Aerospace |
| Lynch, J. | Honeywell |
| Maclellan, A. | Honeywell |
| Marcoux, J. | European Space Agency |
| Martinez, L. | Boeing |
| Masters, D. | Pratt & Whitney Aircraft |
| Matheson, W. | Camber Corp |
| Matthews, P. | Pratt and Whitney Canada Inc |
| Matthews, S. | Avidyne Corp |
| McAnany, I. | Dornier Satellite Systems |
| McCallister, R. | FAA |
| McCormick, F. | Certification Services Inc. |
| McGookey, J. | Smiths Industries |
| McIntyre, G. | FAA, ASY300 |
| McKenna, P. | Smiths Industries Aerospace |
| McKinlay, A. | Booz Allen & Hamilton Inc. |
| McLoughlin, F. | Seagull Technology |
| Meer, M. | Compaq/Digital Computer Corp |
| Melet, V. | DGAC/STNA 3ED |
| Mendez, R. | FAA |
| Meyers, J. | FAA |
| Miles, C. | FAA |
| Moore, P. | Civil Aviation Authority |
| Morgan, S. | MoD (PE) |
| Mortensen, U. | European Space Agency |
| Moyer, F. | Rockwell Collins Avionics |
| Mueller, W. | Bodenseewerk Geratetechnik GmbH (BGT) |
| Muench, M. | Trimble Navigation |
| Mugridge, C. | Defence Evaluation Research Agency |
| Murthi, A. | Strategic Technology Institute |
| Nash, D. | Raytheon Systems |
| Neill, W. | Harris Corporation |
| Nickerson, S. | GE Aircraft Engines |
| Norris, J. | Eldec Corporation |
| Obeid, S. | AlliedSignal Aerospace |
| O'Neil, K. | Advanced Aviation Technology |
| Nogue, J-M. | Aerospatiale |
| Osborg, G. | Normarc AS |
| Ostrom, G. | Allied Signal CAS |
| Paasch, S. | Federal Aviation Administration |
| Paly, G. | SNECMA-ELECMA |
| Pehrson, R. | Boeing |
| Pierce, R. | York Software Engineering |
| Penny, J. | Independent Consultant |
| Perret, B. | EUROCAE |
| Petersen, J. | FAA |
| Pflug, B. | Boeing |
| Phan, T. | FAA, ANM-130L |
| Pilj, G. | Learjet |
| Piper, E. | Boeing - Long Beach Division |
| Porzi, M. | ELMER S.p.A. |
| Pothon, F. | Turbomeca |
| Prisaznuk, P. | AEEC ARINC |
| Pro, B. | Allied Signal EAS |
| Pumfrey, J. | CAA SRG ATS |
| Pyster, A. | FAA |
| Ramsey, M. | Allied Signal EAS |
| Rao, R. | Transport Canada |
| Rebai, K. | |
| Reeve, T. | AVTECH Corp. |
| Reich, A. | Racal Avionics |
| Rhoads, T. | BF Goodrich Aerospace |
| Richardson, M. | Cessna Aircraft Co. |
| Rierson, L. | FAA |
| Rigby, K. | Independent Consultant |
| Rodrigues, S. | Thomson Simulation and Training |
| Rogness, L. | Rockwell Collins |

| | |
|---|---|
| Romanski, G. | Aonix |
| Ronback, J. | Raytheon Systems Canada Ltd |
| Roth, T. | Allied Signal CAS |
| Russo, J-C. | SNECMA |
| Ryburn, R. | FAA, ANM-130L |
| Saderstrom, C. | Flight Dynamics |
| Sahlin, K. | Allied Signal |
| Salmon, P. | Dassault Electronique |
| Sanders, L. | Honeywell |
| Saraceni, P. | FAA |
| Schlickenmaier, H. | NASA |
| Schmitz, D. | Penny and Giles Aerospace |
| Schneider, J-Y. | AIRSYS ATM |
| Schoke, W. | Air Traffic Management USAF |
| Secher, C. | DGAC STNA/2 |
| Seeman, R. | Boeing |
| Severson, M. | Bell Helicopter Textron |
| Shah, J. | Parker-Hannifin ESD |
| Shyne, K. | Boeing |
| Silva, L. | Smiths Industries CSM |
| Singh, R. | FAA AIR-200 |
| Smith, B. | The Soaring Society of America |
| Smith, M. | Ametek Aerospace Products |
| Smith, S. | FAA ASY-300 |
| Sniezek, D. | Lockheed Martin |
| Sonnek, M. | Honeywell |
| Souter, R. | FAA, Wichita ACO |
| Spitzer, C. | Avionicon Inc. |
| Stalford, R. | Litef Gmbh |
| Stallwitz, C. | Raytheon Aircraft |
| Starnes, T. | Cessna Aircraft Company |
| Stolzenthaler, J. | Hamilton Standard/UTC |
| Strachan, I. | AEA Technology |
| Stroup, R. | FAA |
| Struck, W. | FAA |
| Stufflebeam, P. | Raytheon Aircraft Company |
| Tamburro, G. | Registro Aeronautico Italiano |
| Taylor, T. | Parker-Hannifin ESD |
| Temprement, T. | Sextant Avionique |
| Thompson, D. | De Havilland Inc. |
| Thompson, L. | Honeywell |
| Thorvaldsson, M. | Saab Ericsson Space |
| Tochilin, A. | Ashtech Inc |
| Tougas, T. | Airsys ATM Inc |
| Tourret, O. | DGA/SPAé |
| Tripp, L. | Boeing |
| Van Trees, S. | FAA AIR-130 |
| Tutin, I. | BF Goodrich Aerospace |
| Vaudrey, A. | British Embassy |
| Waite, D. | Aerospace Engineering & Research Inc. |
| Wallace, D. | FAA |
| Warren, M. | Flight Dynamics |
| Watkins, R. | FAA/AOS-240 |
| Wendling, B. | Turbomeca |
| Werenowski, M. | Airsys ATM (UK) |
| Wiens, E. | Cessna Aircraft Co |
| Williams, J. | FAA |
| Wolf, R. | Consultant DER |
| Wojnarowski, P. | Boeing - Long Beach Division |
| Yang, J. | |
| Young, W. | Allied Signal Canada Inc |
| Ystanes, B-J. | Kongsberg Defence & Aerospace |
| Zanker, U. | Liebherr Aerospace |

| | |
|---|---|
| Romanski, G. | Aonix |
| Ronback, J. | Raytheon Systems Canada Ltd |
| Roth, T. | Allied Signal CAS |
| Russo, J-C. | SNECMA |
| Ryburn, R. | FAA, ANM-130L |
| Saderstrom, C. | Flight Dynamics |
| Sahlin, K. | Allied Signal |
| Salmon, P. | Dassault Electronique |
| Sanders, L. | Honeywell |
| Saraceni, P. | FAA |
| Schlickenmaier, H. | NASA |
| Schmitz, D. | Penny and Giles Aerospace |
| Schneider, J-Y. | AIRSYS ATM |
| Schoke, W. | Air Traffic Management USAF |
| Secher, C. | DGAC STNA/2 |
| Seeman, R. | Boeing |
| Severson, M. | Bell Helicopter Textron |
| Shah, J. | Parker-Hannifin ESD |
| Shyne, K. | Boeing |
| Silva, L. | Smiths Industries CSM |
| Singh, R. | FAA AIR-200 |
| Smith, B. | The Soaring Society of America |
| Smith, M. | Ametek Aerospace Products |
| Smith, S. | FAA ASY-300 |
| Sniezek, D. | Lockheed Martin |
| Sonnek, M. | Honeywell |
| Souter, R. | FAA, Wichita ACO |
| Spitzer, C. | Avionicon Inc. |
| Stalford, R. | Litef Gmbh |
| Stallwitz, C. | Raytheon Aircraft |
| Starnes, T. | Cessna Aircraft Company |
| Stolzenthaler, J. | Hamilton Standard/UTC |
| Strachan, I. | AEA Technology |
| Stroup, R. | FAA |
| Struck, W. | FAA |
| Stufflebeam, P. | Raytheon Aircraft Company |
| Tamburro, G. | Registro Aeronautico Italiano |
| Taylor, T. | Parker-Hannifin ESD |
| Temprement, T. | Sextant Avionique |
| Thompson, D. | De Havilland Inc. |
| Thompson, L. | Honeywell |
| Thorvaldsson, M. | Saab Ericsson Space |
| Tochilin, A. | Ashtech Inc |
| Tougas, T. | Airsys ATM Inc |
| Tourret, O. | DGA/SPAé |
| Tripp, L. | Boeing |
| Van Trees, S. | FAA AIR-130 |
| Tutin, I. | BF Goodrich Aerospace |
| Vaudrey, A. | British Embassy |
| Waite, D. | Aerospace Engineering & Research Inc. |
| Wallace, D. | FAA |
| Warren, M. | Flight Dynamics |
| Watkins, R. | FAA/AOS-240 |
| Wendling, B. | Turbomeca |
| Werenowski, M. | Airsys ATM (UK) |
| Wiens, E. | Cessna Aircraft Co |
| Williams, J. | FAA |
| Wolf, R. | Consultant DER |
| Wojnarowski, P. | Boeing - Long Beach Division |
| Yang, J. | |
| Young, W. | Allied Signal Canada Inc |
| Ystanes, B-J. | Kongsberg Defence & Aerospace |
| Zanker, U. | Liebherr Aerospace |

APPENDIX C

IMPROVEMENT SUGGESTION FROM

_____

Name: _____ Company Name: _____

Address: _____

City: _____ State, Postal Code, Country: _____

Phone: Date: _____

Document: **DO-178/ED-12 Revision B** Sec: _____ Page: _____ Line: _____

☐ Document error (Format, punctuation, spelling)

☐ Content error

☐ Enhancement or refinement

Rationale (Describe the error or justification for enhancement): _____

_____

_____

_____

Proposed change (attach marked-up text or proposed rewrite): _____

_____

_____

_____

_____

Please provide any general comments for improvement of this document: _____

_____

_____

_____

_____

_____

Return completed form to:

**OR**

**RTCA**
**Attention: DO-178B**
**1140 Connecticut Ave. NW Suite 1020**
**Washington, D.C. 20036 USA**

**EUROCAE**
**Attention: Secretary General**
**17 rue Hamelin**
**75783 Paris Cedex 16, FRANCE**

APPENDIX C

IMPROVEMENT SUGGESTION FROM

_____

Name: _____ Company Name: _____

Address: _____

City: _____ State, Postal Code, Country: _____

Phone: Date: _____

Document: **DO-178/ED-12 Revision B** Sec: _____ Page: _____ Line: _____

☐ Document error (Format, punctuation, spelling)

☐ Content error

☐ Enhancement or refinement

Rationale (Describe the error or justification for enhancement): _____

_____

_____

_____

Proposed change (attach marked-up text or proposed rewrite): _____

_____

_____

_____

_____

Please provide any general comments for improvement of this document: _____

_____

_____

_____

_____

_____

Return completed form to:

**OR**

**RTCA**
**Attention: DO-178B**
**1140 Connecticut Ave. NW Suite 1020**
**Washington, D.C. 20036 USA**

**EUROCAE**
**Attention: Secretary General**
**17 rue Hamelin**
**75783 Paris Cedex 16, FRANCE**

- EUROCAE -

**THE EUROPEAN ORGANISATION FOR CIVIL AVIATION EQUIPMENT**

**ORGANISATION EUROPEENNE POUR L'EQUIPEMENT DE L'AVIATION CIVILE**

EUR 221-99/SECR-86 Paris, October 19th, 1999

DOCUMENT ED-12B / DO-178B

# SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION

AMENDMENT No 1

Note: These amendments have also been published in EUROCAE document ED-94 "First Annual Report For Clarification of ED-12B".

Section 2, § 2.3.2. Replace "specify" by "may specify".

The second sentence of the second paragraph will now read:
"System requirements may specify a hardware configuration which provides for the execution of multiple version dissimilar software."

Section 2, § 2.3.3c Replace "failure condition" by "failure".

This sentence will now read:
"c. Independence of Function and Monitor: The monitor and protective mechanism are not rendered inoperative by the same failure that causes the hazard."

Section 6, § 6.3.2c Replace "requirements" by "low-level requirements".

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

Section 6, § 6.3.3f Delete "or isolated".

This sentence will now read:
"f. Partitioning integrity: The objective is to ensure that partitioning breaches are prevented."

- EUROCAE -

**THE EUROPEAN ORGANISATION FOR CIVIL AVIATION EQUIPMENT**

**ORGANISATION EUROPEENNE POUR L'EQUIPEMENT DE L'AVIATION CIVILE**

EUR 264-99/SECR-102 Paris, 19 octobre 1999

DOCUMENT ED-12B / DO-178B

# CONSIDERATIONS SUR LE LOGICIEL EN VUE DE LA CERTIFICATION DES SYSTEMES ET EQUIPEMENTS DE BORDS

AMENDEMENT No 1

Note: Cet amendement est aussi publié dans le document EUROCAE ED-94 "First Annual Report For Clarification of ED-12B".

Section 2, § 2.3.2. Remplacer « définissent » par « peuvent définir ».

La deuxième phrase du second paragraphe devient :
« Les spécifications du système peuvent définir une configuration matérielle qui assure d'un logiciel à versions multiple dissimilaires »

Section 2, § 2.3.3c Remplacer « conditions de pannes» par « pannes».

La phrase devient :
"c. Indépendance de la fonction et du dispositif de surveillance : Le dispositif de surveillance et le mécanisme de protection ne sont pas rendus inopérants par la même panne qui cause le danger."

Section 6, § 6.3.2c Remplacer « exigences » par «exigences de bas niveau ».

La phrase devient :
"c. Compatibilité avec la machine cible : L'objectif est de s'assurer qu'il n'existe pas de conflits entre les exigences logicielles de bas niveau et les caractéristiques matérielles et logicielles de la machine cible, en particulier en ce qui concerne l'utilisation de ressources (telles qu 'un chargement de bus), les temps de réponse du système, et les dispositifs d'entrées et de sortie.

Section 6, § 6.3.3f Supprimer « ou isolées »

La phrase devient :
"f. Intégrité du partitionnement : L'objectif est de s'assurer que les violations du partitionnement sont évitées.

Section 6, § 6.4 Delete "PROCESS" in the title of this paragraph.

The title will now read:
"SOFTWARE TESTING".

Section 6, § 6.4 In the first line of the second paragraph, replace "process" by "activities".

This sentence will now read:
"Figure 6-1 is a diagram of the software testing activities."

Section 6, Figure 6-1 In the title of Figure-1, replace "process" by "activities".

The title will now read:
"SOFTWARE TESTING ACTIVITIES"

Annex A, Table A-4 The reference number for objective 9 should be changed from"6.3.2b" to "6.3.3b".

Section 6, § 6.4 Dans la première phrase du second paragraphe remplacer « tests » par « activités de test ».

La phrase devient :
"Figure 6-1 représente un diagramme des activités de test d'un logiciel.

Section 6, Figure 6-1 Dans le titre de la figure 6-1 remplacer « test » par « activité de test »
La phrase devient :

"ACTIVITES DE TEST DU LOGICIEL"