



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

## SOFTWARE INTEGRITY ASSURANCE CONSIDERATIONS FOR COMMUNICATION, NAVIGATION, SURVEILLANCE AND AIR TRAFFIC MANAGEMENT (CNS/ATM) SYSTEMS

This document is the exclusive intellectual and commercial property of EUROCAE.

It is presently commercialised by EUROCAE.

**This electronic copy is delivered to your company/organisation for internal use exclusively.**

In no case it may be re-sold, or hired, lent or exchanged outside your company.

### **ED-109A**

January 2012

# SOFTWARE INTEGRITY ASSURANCE CONSIDERATIONS FOR COMMUNICATION, NAVIGATION, SURVEILLANCE AND AIR TRAFFIC MANAGEMENT (CNS/ATM) SYSTEMS

This document is the exclusive intellectual and commercial property of EUROCAE.

It is presently commercialised by EUROCAE.

**This electronic copy is delivered to your company/organisation for internal use exclusively.**

In no case it may be re-sold, or hired, lent or exchanged outside your company.

**ED-109A**

January 2012

## FOREWORD

1. This document was prepared jointly by EUROCAE Working Group #71 (WG-71) and RTCA Special Committee #205 (SC-205), was approved by the Council of EUROCAE on 9 January 2012.
2. EUROCAE is an international non-profit making organisation in Europe. Membership is open to manufacturers and users of equipment for aeronautics, trade associations, national civil aviation administrations, and, under certain conditions, non-European organisations. Its work programme is principally directed to the preparation of performance specifications and guidance documents for civil aviation equipment, for adoption and use at European and world-wide levels.
3. The findings of EUROCAE are resolved after discussion amongst Members of EUROCAE and in collaboration with RTCA Inc, Washington D.C, through their appropriate committees.
4. EUROCAE performance specifications and other documents 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

102, rue Etienne Dolet

92240 MALAKOFF

France

Tel: 33 1 40 92 79 30

Fax: 33 1 46 55 62 65

Email: [eurocae@eurocae.net](mailto:eurocae@eurocae.net)

Website: [www.eurocae.net](http://www.eurocae.net)

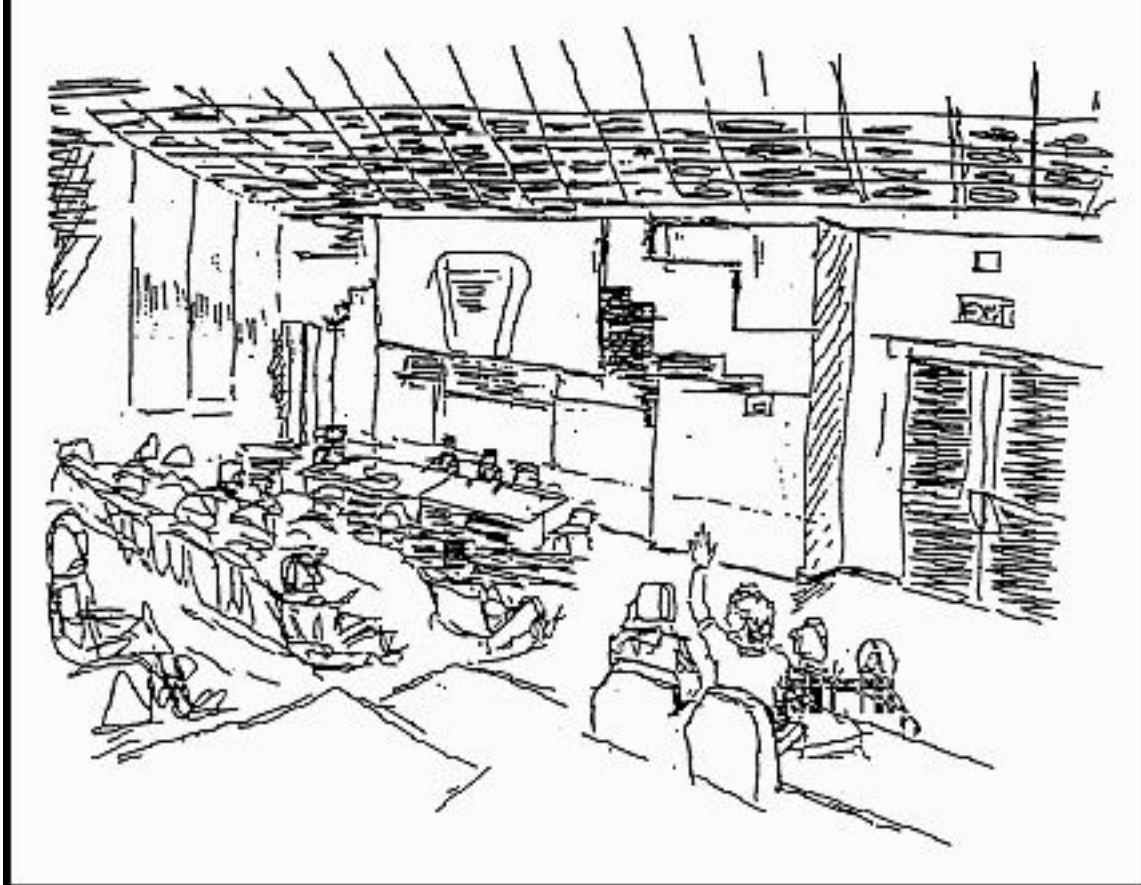


Illustration provided by Pat Neilan, UK CAA

**CONSENSUS** n. Collective opinion or concord; general agreement or accord. [Latin, from consentire, to agree]

## TABLE OF CONTENTS

|       |                                                                          |    |
|-------|--------------------------------------------------------------------------|----|
| 1.0   | INTRODUCTION .....                                                       | 1  |
| 1.1   | Purpose .....                                                            | 1  |
| 1.2   | Scope .....                                                              | 1  |
| 1.3   | Relationship to Other Documents .....                                    | 2  |
| 1.4   | How to Use This Document.....                                            | 2  |
| 1.5   | Document Overview .....                                                  | 4  |
| 2.0   | SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT .....                    | 5  |
| 2.1   | System Requirements Allocation to Software .....                         | 5  |
| 2.2   | Information Flow Between System and Software Life Cycle Processes        | 6  |
| 2.2.1 | Information Flow from System Processes to Software Processes             | 8  |
| 2.2.2 | Information Flow from Software Processes to System Processes             | 8  |
| 2.2.3 | Information Flow between Software Processes and Hardware Processes ..... | 9  |
| 2.3   | System Safety Assessment Process and Assurance Level.....                | 9  |
| 2.3.1 | Relationship between Software Errors and Failure Conditions              | 10 |
| 2.3.2 | Assurance Level Definitions.....                                         | 10 |
| 2.3.3 | Assurance Level Definitions.....                                         | 11 |
| 2.3.4 | Assurance Level Determination .....                                      | 12 |
| 2.4   | Architectural Considerations .....                                       | 12 |
| 2.4.1 | Partitioning .....                                                       | 13 |
| 2.4.2 | Multiple-Version Dissimilar Software .....                               | 13 |
| 2.4.3 | Safety Monitoring .....                                                  | 14 |
| 2.5   | Additional System Considerations.....                                    | 14 |
| 2.5.1 | System Communication .....                                               | 14 |
| 2.5.2 | Security .....                                                           | 14 |
| 2.5.3 | Adaptability.....                                                        | 14 |
| 2.5.4 | Cutover (Hot Swapping).....                                              | 15 |
| 2.5.5 | Post-Development Life Cycle.....                                         | 15 |
| 2.6   | Software Considerations in System Life Cycle Processes.....              | 15 |
| 2.6.1 | Adaptation Data Items .....                                              | 15 |
| 2.6.2 | User-Modifiable Software.....                                            | 16 |
| 2.6.3 | Commercial-Off-The-Shelf Software.....                                   | 17 |
| 2.6.4 | Option-Selectable Software .....                                         | 17 |
| 2.6.5 | Field-Loadable Software .....                                            | 17 |
| 2.6.6 | Software Considerations in System Verification .....                     | 18 |
| 2.7   | System Considerations in Software Life Cycle Processes.....              | 18 |

|       |                                                                      |    |
|-------|----------------------------------------------------------------------|----|
| 3.0   | SOFTWARE LIFE CYCLE .....                                            | 19 |
| 3.1   | Software Life Cycle Processes.....                                   | 19 |
| 3.2   | Software Life Cycle Definition .....                                 | 19 |
| 3.3   | Transition Criteria Between Processes .....                          | 20 |
| 4.0   | SOFTWARE PLANNING PROCESS .....                                      | 21 |
| 4.1   | Software Planning Process Objectives .....                           | 21 |
| 4.2   | Software Planning Process Activities .....                           | 21 |
| 4.3   | Software Plans .....                                                 | 22 |
| 4.4   | Software Life Cycle Environment Planning .....                       | 23 |
| 4.4.1 | Software Development Environment .....                               | 23 |
| 4.4.2 | Language and Compiler Considerations .....                           | 24 |
| 4.4.3 | Software Test Environment.....                                       | 24 |
| 4.5   | Software Development Standards.....                                  | 24 |
| 4.6   | Review of the Software Planning Process .....                        | 25 |
| 5.0   | SOFTWARE DEVELOPMENT PROCESSES.....                                  | 25 |
| 5.1   | Software Requirements Process.....                                   | 27 |
| 5.1.1 | Software Requirements Process Objectives .....                       | 27 |
| 5.1.2 | Software Requirements Process Activities .....                       | 27 |
| 5.2   | Software Design Process .....                                        | 28 |
| 5.2.1 | Software Design Process Objectives.....                              | 28 |
| 5.2.2 | Software Design Process Activities .....                             | 28 |
| 5.2.3 | Designing for User-Modifiable Software .....                         | 29 |
| 5.2.4 | Designing for Deactivated Code .....                                 | 29 |
| 5.3   | Software Coding Process.....                                         | 29 |
| 5.3.1 | Software Coding Process Objectives.....                              | 29 |
| 5.3.2 | Software Coding Process Activities .....                             | 29 |
| 5.4   | Integration Process.....                                             | 30 |
| 5.4.1 | Integration Process Objectives .....                                 | 30 |
| 5.4.2 | Integration Process Activities.....                                  | 30 |
| 5.5   | Software Development Process Traceability .....                      | 31 |
| 6.0   | SOFTWARE VERIFICATION PROCESS .....                                  | 32 |
| 6.1   | Purpose of Software Verification.....                                | 32 |
| 6.2   | Overview of Software Verification Process Activities .....           | 33 |
| 6.3   | Software Reviews and Analyses .....                                  | 34 |
| 6.3.1 | Reviews and Analyses of High-Level Requirements .....                | 34 |
| 6.3.2 | Reviews and Analyses of Low-Level Requirements.....                  | 34 |
| 6.3.3 | Reviews and Analyses of Software Architecture .....                  | 35 |
| 6.3.4 | Reviews and Analyses of Source Code.....                             | 35 |
| 6.3.5 | Reviews and Analyses of the Outputs of the Integration Process ..... | 36 |

|       |                                                                       |    |
|-------|-----------------------------------------------------------------------|----|
| 6.4   | Software Testing.....                                                 | 36 |
| 6.4.1 | Test Environment.....                                                 | 38 |
| 6.4.2 | Requirements-Based Test Selection .....                               | 39 |
| 6.4.3 | Requirements-Based Testing Methods .....                              | 40 |
| 6.4.4 | Test Coverage Analysis .....                                          | 41 |
| 6.4.5 | Reviews and Analyses of Test Cases, Procedures, and Results .....     | 43 |
| 6.5   | Software Verification Process Traceability .....                      | 43 |
| 6.6   | Verification of Adaptation Data Items .....                           | 43 |
| 7.0   | SOFTWARE CONFIGURATION MANAGEMENT PROCESS .....                       | 45 |
| 7.1   | Software Configuration Management Process Objectives.....             | 45 |
| 7.2   | Software Configuration Management Process Activities .....            | 46 |
| 7.2.1 | Configuration Identification .....                                    | 46 |
| 7.2.2 | Baselines and Traceability .....                                      | 46 |
| 7.2.3 | Problem Reporting, Tracking, and Corrective Action.....               | 47 |
| 7.2.4 | Change Control .....                                                  | 47 |
| 7.2.5 | Change Review.....                                                    | 48 |
| 7.2.6 | Configuration Status Accounting.....                                  | 48 |
| 7.2.7 | Archive, Retrieval, and Release.....                                  | 48 |
| 7.3   | Data Control Categories .....                                         | 49 |
| 7.4   | Software Load Control.....                                            | 50 |
| 7.5   | Software Life Cycle Environment Control .....                         | 50 |
| 8.0   | SOFTWARE QUALITY ASSURANCE PROCESS.....                               | 51 |
| 8.1   | Software Quality Assurance Process Objectives .....                   | 51 |
| 8.2   | Software Quality Assurance Process Activities.....                    | 51 |
| 8.3   | Software Conformity Review .....                                      | 52 |
| 9.0   | APPROVAL LIAISON PROCESS.....                                         | 54 |
| 9.1   | Means of Compliance and Planning.....                                 | 54 |
| 9.2   | Compliance Substantiation.....                                        | 54 |
| 9.3   | Minimum Software Life Cycle Data Submitted to Approval Authority .... | 55 |
| 9.4   | Software Life Cycle Data Related to the Approval Process .....        | 55 |
| 10.0  | OVERVIEW OF CNS/ATM SYSTEM APPROVAL PROCESS .....                     | 56 |
| 10.1  | Approval Basis.....                                                   | 56 |
| 10.2  | Software Aspects of Approval .....                                    | 56 |
| 10.3  | Compliance Determination .....                                        | 56 |
| 11.0  | SOFTWARE LIFE CYCLE DATA .....                                        | 57 |
| 11.1  | Plan for Software Aspects of Approval.....                            | 58 |
| 11.2  | Software Development Plan .....                                       | 59 |
| 11.3  | Software Verification Plan .....                                      | 59 |
| 11.4  | Software Configuration Management Plan .....                          | 60 |
| 11.5  | Software Quality Assurance Plan.....                                  | 61 |

|        |                                                                           |    |
|--------|---------------------------------------------------------------------------|----|
| 11.6   | Software Requirements Standards .....                                     | 61 |
| 11.7   | Software Design Standards.....                                            | 62 |
| 11.8   | Software Code Standards .....                                             | 62 |
| 11.9   | Software Requirements Data .....                                          | 62 |
| 11.10  | Design Description .....                                                  | 63 |
| 11.11  | Source Code.....                                                          | 63 |
| 11.12  | Executable Object Code.....                                               | 63 |
| 11.13  | Software Verification Cases and Procedures.....                           | 64 |
| 11.14  | Software Verification Results .....                                       | 64 |
| 11.15  | Software Life Cycle Environment Configuration Index.....                  | 64 |
| 11.16  | Software Configuration Index .....                                        | 65 |
| 11.17  | Problem Reports.....                                                      | 65 |
| 11.18  | Software Configuration Management Records .....                           | 65 |
| 11.19  | Software Quality Assurance Records.....                                   | 66 |
| 11.20  | Software Accomplishment Summary .....                                     | 66 |
| 11.21  | Trace Data.....                                                           | 67 |
| 11.22  | Adaptation Data Item File.....                                            | 67 |
| 12.0   | ADDITIONAL CONSIDERATIONS.....                                            | 68 |
| 12.1   | Use of Previously Developed Software .....                                | 68 |
| 12.1.1 | Modifications to Previously Developed Software.....                       | 68 |
| 12.1.2 | Reuse of Previously Approved Software in a CNS/ATM System .....           | 68 |
| 12.1.3 | Change of Application or Development Environment.....                     | 69 |
| 12.1.4 | Upgrading a Development Baseline .....                                    | 70 |
| 12.1.5 | Software Configuration Management Considerations .....                    | 70 |
| 12.1.6 | Software Quality Assurance Considerations .....                           | 71 |
| 12.2   | Tool Qualification.....                                                   | 71 |
| 12.2.1 | Determining if Tool Qualification is Needed.....                          | 71 |
| 12.2.2 | Determining the Tool Qualification Level .....                            | 71 |
| 12.2.3 | Tool Qualification Process .....                                          | 72 |
| 12.3   | Alternative Methods.....                                                  | 72 |
| 12.3.1 | Exhaustive Input Testing.....                                             | 72 |
| 12.3.2 | Considerations for Multiple-Version Dissimilar Software Verification..... | 73 |
| 12.3.3 | Software Reliability Models.....                                          | 75 |
| 12.3.4 | Service Experience .....                                                  | 75 |
| 12.4   | Commercial Off-The-Shelf Software .....                                   | 79 |
| 12.4.1 | Introduction .....                                                        | 79 |
| 12.4.2 | System Aspects of COTS Software.....                                      | 81 |
| 12.4.3 | COTS Software Planning Process.....                                       | 81 |
| 12.4.4 | COTS Software Acquisition Process .....                                   | 83 |
| 12.4.5 | COTS Software Verification Process .....                                  | 86 |

|            |                                                                                         |     |
|------------|-----------------------------------------------------------------------------------------|-----|
| 12.4.6     | COTS Software Configuration Management Process .....                                    | 86  |
| 12.4.7     | COTS Software Quality Assurance Process .....                                           | 87  |
| 12.4.8     | Software Life Cycle Data .....                                                          | 88  |
| 12.4.9     | Changes to COTS from an Earlier Baseline .....                                          | 89  |
| 12.4.10    | Additional Process Objectives and Outputs by Assurance Level for<br>COTS Software ..... | 89  |
| 12.4.11    | Alternative Methods for Providing Assurance of COTS Software                            | 93  |
| ANNEX A    | PROCESS OBJECTIVES AND OUTPUTS BY ASSURANCE LEVEL .....                                 | 103 |
| ANNEX B    | ACRONYMS AND GLOSSARY OF TERMS .....                                                    | 114 |
| APPENDIX A | BACKGROUND OF ED-109/DO-278 DOCUMENT .....                                              | 125 |
| APPENDIX B | COMMITTEE MEMBERSHIP .....                                                              | 132 |

### LIST OF FIGURES

|             |                                                                               |    |
|-------------|-------------------------------------------------------------------------------|----|
| FIGURE 1-1  | DOCUMENT OVERVIEW.....                                                        | 4  |
| FIGURE 2-1  | INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES.....        | 7  |
| FIGURE 2-2  | SEQUENCE OF EVENTS FOR SOFTWARE ERROR LEADING TO A FAILURE CONDITION.....     | 10 |
| FIGURE 3-1  | EXAMPLE OF A SOFTWARE PROJECT USING FOUR DIFFERENT DEVELOPMENT SEQUENCES..... | 20 |
| FIGURE 6-1  | SOFTWARE TESTING ACTIVITIES .....                                             | 38 |
| FIGURE 12-1 | REQUIREMENTS INTERSECTION.....                                                | 84 |

### LIST OF TABLES

|            |                                                                           |     |
|------------|---------------------------------------------------------------------------|-----|
| TABLE 2-1  | EXAMPLE FAILURE CONDITION CATEGORY DESCRIPTIONS.....                      | 11  |
| TABLE 2-2  | CNS/ATM TO AIRBORNE LEVEL ASSOCIATION.....                                | 12  |
| TABLE 7-1  | SCM PROCESS ACTIVITIES ASSOCIATED WITH CC1 AND CC2 DATA .....             | 49  |
| TABLE 12-1 | TOOL QUALIFICATION LEVEL DETERMINATION .....                              | 72  |
| TABLE 12-2 | COTS SOFTWARE PLANNING PROCESS OBJECTIVES.....                            | 90  |
| TABLE 12-3 | COTS SOFTWARE ACQUISITION PROCESS OBJECTIVES.....                         | 91  |
| TABLE 12-4 | COTS SOFTWARE VERIFICATION PROCESS OBJECTIVES.....                        | 92  |
| TABLE 12-5 | COTS SOFTWARE CONFIGURATION MANAGEMENT PROCESS OBJECTIVES .....           | 92  |
| TABLE A-1  | SOFTWARE PLANNING PROCESS .....                                           | 104 |
| TABLE A-2  | SOFTWARE DEVELOPMENT PROCESSES.....                                       | 105 |
| TABLE A-3  | VERIFICATION OF OUTPUTS OF SOFTWARE REQUIREMENTS PROCESS .....            | 106 |
| TABLE A-4  | VERIFICATION OF OUTPUTS OF SOFTWARE DESIGN PROCESS .....                  | 107 |
| TABLE A-5  | VERIFICATION OF OUTPUTS OF SOFTWARE CODING AND INTEGRATION PROCESSES..... | 108 |
| TABLE A-6  | TESTING OF OUTPUTS OF INTEGRATION PROCESS.....                            | 109 |
| TABLE A-7  | VERIFICATION OF VERIFICATION PROCESS RESULTS.....                         | 110 |
| TABLE A-8  | SOFTWARE CONFIGURATION MANAGEMENT PROCESS .....                           | 111 |
| TABLE A-9  | SOFTWARE QUALITY ASSURANCE PROCESS.....                                   | 112 |
| TABLE A-10 | SOFTWARE APPROVAL PROCESS .....                                           | 113 |

## CHAPTER 1

### INTRODUCTION

The implementation of Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) systems has resulted in increased interdependence of systems providing Air Traffic Services (ATS) and systems onboard aircraft. CNS/ATM systems can include ground, airborne, and space-based elements. In order for these systems to perform their intended function while providing an acceptable level of safety, there is a need to define consistent and/or equivalent means of providing integrity assurance for the software (SW) in these systems. ED-109, "Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems," was written to satisfy the non-airborne (that is, ground and space) aspects of this need. Appendix A provides the background of this document.

#### 1.1 PURPOSE

The purpose of this document is to provide guidance for the production of non-airborne software for CNS/ATM systems and equipment that performs its intended function with a level of confidence in safety that complies with approval requirements. This guidance includes:

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

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

#### 1.2 SCOPE

This document discusses those aspects of approval that pertain to the production of software for CNS/ATM systems. 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 approval process. A complete description of the system life cycle processes, including the system safety assessment and validation processes, or the approval process is not intended.

The guidance contained in this document does not define or imply the level of involvement of an approval authority in an approval process. To understand approval authority involvement, the applicant should refer to applicable regulations and guidance material issued by the relevant approval authority.

This document does not attempt to define firmware. Firmware such as embedded microcode or libraries should be classified as hardware or software and addressed by the applicable processes. This document assumes that during the system definition, functions have been allocated to either software or hardware. Other documents exist that provide guidance for development assurance for functions that are allocated to implementation in hardware. This document provides guidance for functions that are allocated to software.

**NOTE:** This allows an efficient method of implementation and development assurance to be determined at the time the system is specified and functions allocated. All parties should agree with this system decision at the time the allocation is made.

Matters concerning the structure of the applicant's organization, the commercial relationships between the applicant and its suppliers, and personnel qualification criteria are beyond the scope of this document.

### 1.3 RELATIONSHIP TO OTHER DOCUMENTS

During the development of this document, various national and international software integrity assurance standards were reviewed. While the means of compliance may differ, no incompatibilities were identified with these standards. These other standards are available through national and international agencies. 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 in addition to this document.

It is recognized that projects may be obliged, through contract or other means, to comply with additional standards as applied by, for example, an air navigation service provider. Such standards may be derived from general standards produced or adopted by the manufacturer for its activities. Such standards should be considered by the planning process and considered, as appropriate, when applying supplier oversight.

### 1.4 HOW TO USE THIS DOCUMENT

The following points should be noted when using this document:

- a. This document is intended to be used by the international CNS/ATM community. To aid such use, references to specific national regulations and procedures are minimized. Instead, generic terms are used. For example, the term "approval authority" is used to denote the organization or person granting approval.
- b. This document recognizes that the guidance herein is not mandated by law, but represents a consensus of the CNS/ATM 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.
- c. If an applicant adopts this document as a means of compliance, the applicant should satisfy all applicable objectives. This document should apply to the applicant and any of its suppliers who are involved with any of the software life cycle processes or the outputs of those processes described herein. The applicant is responsible for oversight of all of its suppliers.
- d. The applicant should plan a set of activities that satisfy the objectives. This document describes activities for achieving those objectives. The applicant may plan and, subject to the approval of the approval authority, adopt alternative activities to those described in this document. The applicant may also plan and conduct additional activities that are determined to be necessary.
- e. The applicant should address any additional considerations in its software plans and standards.
- f. The applicant should perform the planned activities and provide evidence as indicated in section 11 to substantiate that the objectives have been satisfied.
- g. 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 provides a description of the software life cycle and section 10 an overview of the CNS/ATM system approval process.

- h. Section 11 contains the data generally produced to aid the software aspects of the approval 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).
- i. Section 12 discusses additional considerations including guidance for the use of previously developed software, tool qualification, and the use of alternative methods to those described in sections 2 through 11. Section 12 may not apply to every project.
- j. Annex A specifies the applicability of the objectives, activities, and software life cycle data for each assurance level as well as the variation in the independence and control categories for each assurance level. In order to fully understand the guidance, the full body of this document should be considered.
- k. In cases where examples are used to indicate how the guidance might be applied, either graphically or through narrative, the examples are not to be interpreted as the preferred method. In these cases, the examples are considered supporting information.
- l. A list of items does not imply the list is all-inclusive.
- m. Notes in this document are supporting information used to provide explanatory material, emphasize a point, or draw attention to related items that are not entirely within context.
- n. Major sections are numbered as X.0 throughout this document. It should be noted that references to an entire section are identified as “section X”; whereas, references to the content between section headers X.0 and X.1 are referenced as “section X.0”.
- o. One or more supplements to this document exist and extend the guidance in this document to a specific technique. Supplements are used in conjunction with this document and may be used in conjunction with one another. Unless alternatives are used (see 1.4.i), if a supplement exists for a specific technique, the supplement should be used to add, delete, or otherwise modify objectives, activities, explanatory text, and software life cycle data in this document to address that technique, as defined appropriately in each supplement. It is the responsibility of the applicant to ensure that the supplement’s use is acceptable to the appropriate approval authority. As part of the software planning process, the applicant should review all potentially relevant supplements and identify those that will be used. The information in supplements should be used with and in the same way as this document. Annex C of each supplement identifies how the objectives of this document are revised relative to the specific technique addressed by the supplement.
- p. Compliance is achieved when all applicable objectives have been satisfied by performing all planned activities and capturing the related evidence.

**NOTE:** Unlike its predecessor (ED-109), this document is a stand-alone document that is not dependent on ED-12C when being used.

## 1.5 DOCUMENT OVERVIEW

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

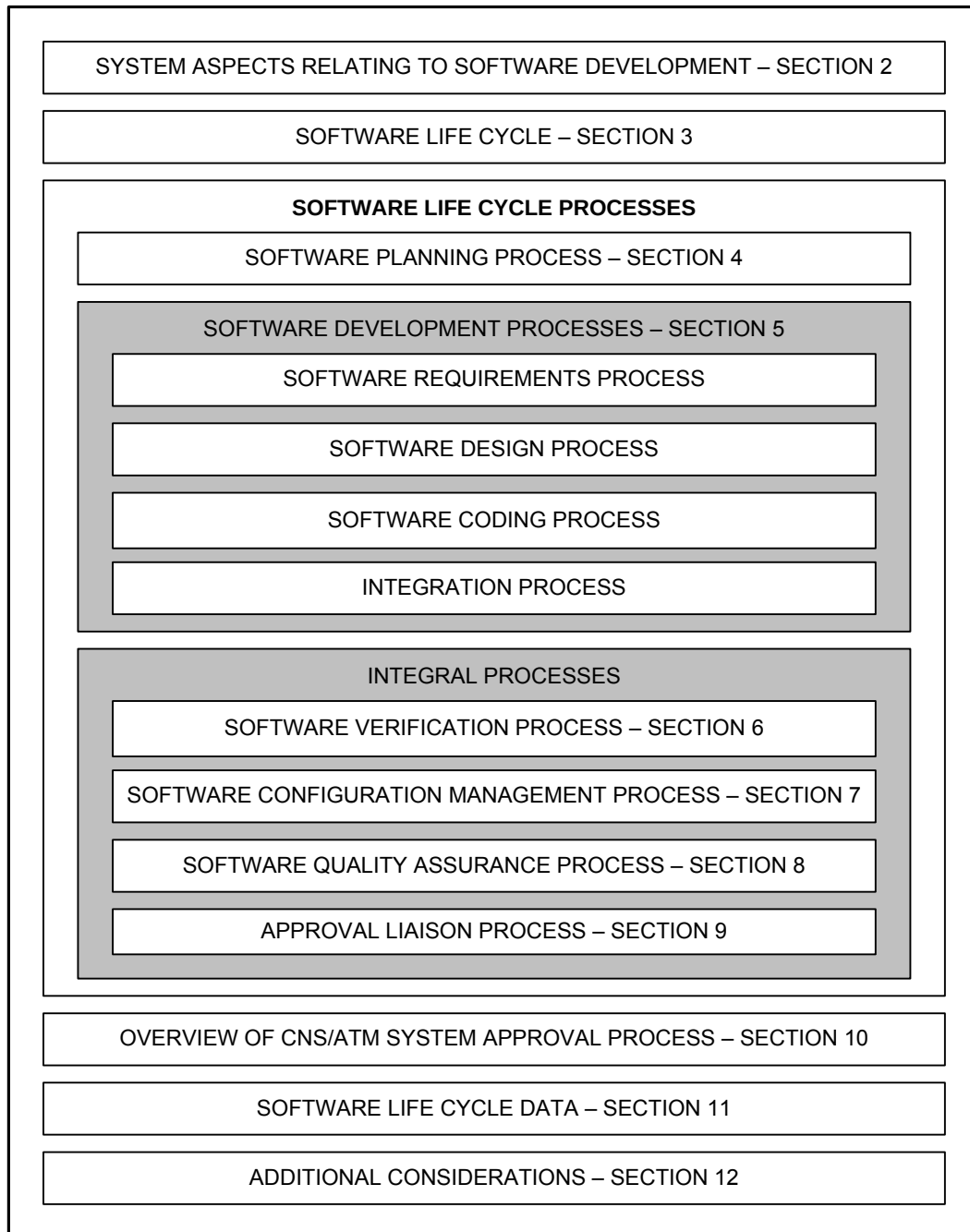


**FIGURE 1-1: DOCUMENT OVERVIEW**

## CHAPTER 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. System life cycle processes can be found in other industry documents.

Discussed in this section are:

- System requirements allocation to software (see 2.1).
- The information flow between the system and software life cycle processes and between the software and hardware life cycle processes (see 2.2).
- The system safety assessment process, assurance level definitions and assurance level determination (see 2.3).
- Architectural considerations (see 2.4).
- Additional system considerations (see 2.5).
- Software considerations in system life cycle processes (see 2.6).

The term “system” in the context of this document refers to CNS/ATM systems and equipment only, not to the wider definition of a system that might include operators, operational procedures, etc.

#### 2.1

#### SYSTEM REQUIREMENTS ALLOCATION TO SOFTWARE

As part of the system life cycle processes, system requirements are developed from the system operational requirements and other considerations such as safety-related, security, and performance requirements. The safety-related requirements result from the system safety assessment process, and may include functional, integrity, and reliability requirements, as well as design constraints.

Within the safety assessment process, safety-related requirements are defined to ensure the integrity of the system by specifying the desired immunity from, and system responses to, failure conditions. These requirements are identified for hardware and software to preclude or limit the effects of faults, and may provide fault detection, fault tolerance, fault removal and fault avoidance.

The system processes are responsible for the refinement and allocation of system requirements to hardware and/or software as determined by the system architecture.

System requirements allocated to software, including safety-related requirements, are developed and refined into software requirements that are verified by the software verification process activities. These requirements and the associated verification should establish that the software performs its intended functions under any foreseeable operating condition. System requirements allocated to software may include:

- a. Functional and operational requirements.
- b. Interface requirements.
- c. Performance requirements.
- d. Safety-related requirements, including safety strategies, design constraints and design methods, such as, partitioning, dissimilarity, redundancy, or safety monitoring. In cases where the system is a component of another system, the requirements and failure conditions for that other system may also form part of the system requirements allocated to software.
- e. Security requirements.

- f. Maintenance requirements.
- g. Approval requirements, including any applicable approval authority regulations, issue papers etc.
- h. Additional requirements needed to aid the system life cycle processes.

## **2.2 INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES**

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

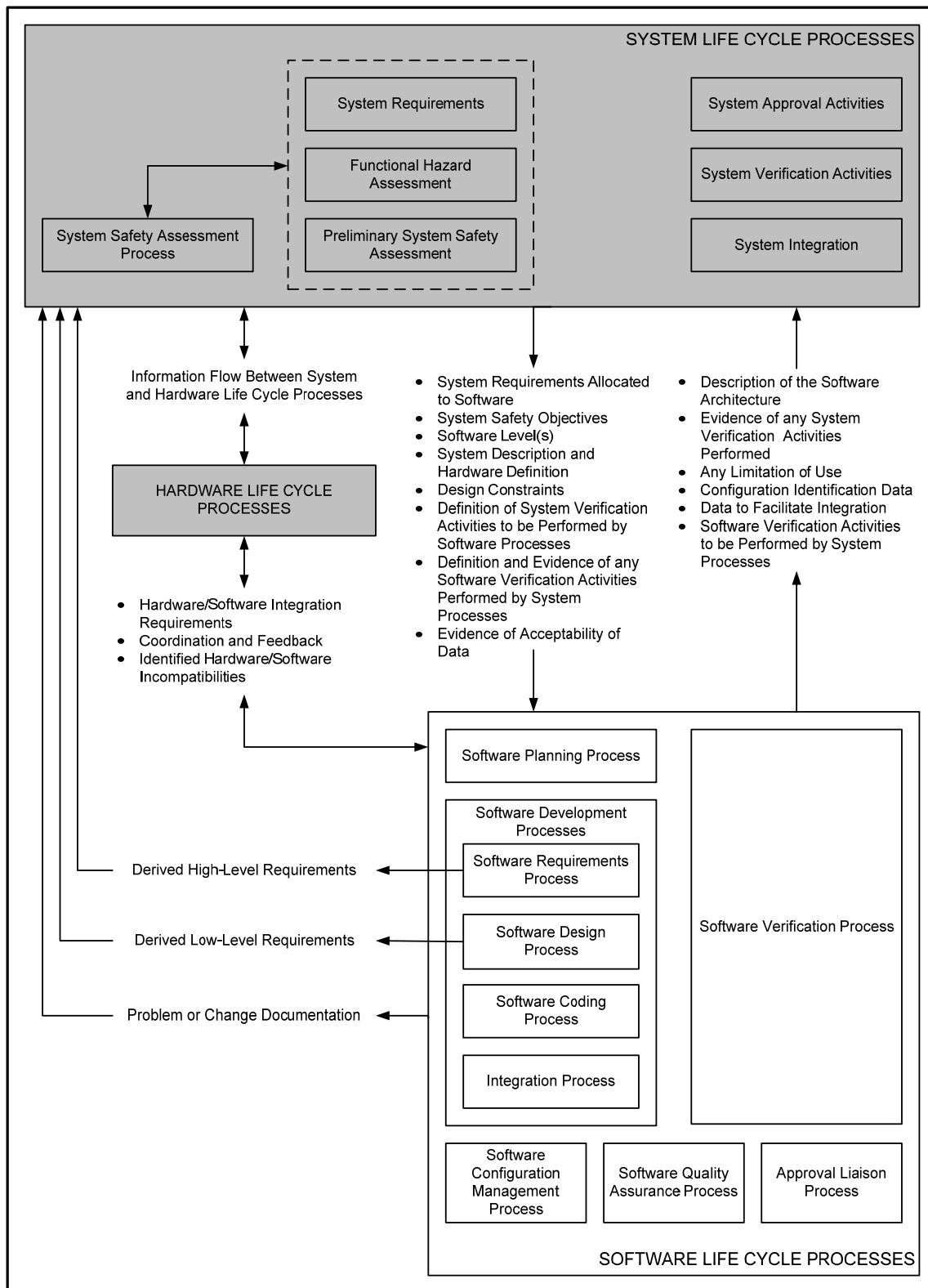


**FIGURE 2-1: INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES**

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

The following data is passed to the software life cycle processes by the system processes either as part of the requirements allocation or during the development life cycle:

- a. System requirements allocated to software.
- b. System safety objectives.
- c. Assurance level for software components and a description of associated failure condition(s), if applicable.
- d. System description and hardware definition.
- e. Design constraints, including external interfaces, partitioning requirements etc.
- f. Details of any system activities proposed to be performed as part of the software life cycle. Note that system requirement validation is not usually part of the software life cycle processes. The system life cycle processes are responsible for assuring any system activities proposed to be performed as part of the software life cycle.
- g. Evidence of the acceptability, or otherwise, of any data provided by the software processes on which any activity has been conducted by the system processes. Examples of such activity are the system processes' evaluations of:
  1. Derived requirements provided by the software processes to determine if there is any impact on the system safety assessment and system requirements.
  2. Issues raised by the software processes with respect to the clarification or correction of system requirements allocated to software.

Evidence of software verification activities performed by the system life cycle processes, if any.

Any evidence provided by the system processes (see 2.2.1.f and 2.2.1.g) should be considered by the software processes to be Software Verification Results (see 11.14).

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

The software life cycle processes analyze the system requirements allocated to software as part of the software requirements process. If such an analysis identifies any system requirements as inadequate or incorrect, the software life cycle processes should capture the issues and refer them to the system processes for resolution. Furthermore, as the software design and implementation evolves, details are added and modifications made that may affect system safety assessment and system requirements.

To aid the evaluation of the evolving design and changes to the design, the software life cycle processes should make data available to the system processes including the system safety assessment process. This data will facilitate analyses and evaluations to establish the effect on the system safety assessment and system requirements. It may be advantageous for such analyses and evaluations to be performed jointly by the systems and software processes. Such data includes:

- a. Details of derived requirements created during the software life cycle processes.
- b. A description of the software architecture, including software partitioning.
- c. Evidence of system activities performed by the software life cycle processes, if any.
- d. Problem or change documentation, including problems identified in the system requirements allocated to software and identified incompatibilities between the hardware and the software.

- e. Any limitations of use.
- f. Configuration identification and any configuration status constraints.
- g. Performance, timing, and accuracy characteristics.
- h. Data to facilitate integration of the software into the system.
- i. Details of software verification activities proposed to be performed during system verification, if any.

### **2.2.3 Information Flow between Software Processes and Hardware Processes**

Data is passed between the software life cycle processes and the hardware life cycle processes either as part of the system requirements allocation or during the development life cycle. Such data includes:

- a. All requirements, including derived requirements, needed for hardware/software integration, such as definition of protocols, timing constraints, and addressing schemes for the interface between hardware and software.
- b. Instances where hardware and software verification activities require coordination.
- c. Identified incompatibilities between the hardware and the software.

## **2.3 SYSTEM SAFETY ASSESSMENT PROCESS AND ASSURANCE LEVEL**

This section provides a brief introduction to how the assurance level for software components is determined and how architectural considerations may influence the allocation of an assurance level. It is not the intent of this document to prescribe how these activities are performed; this has to be established and performed as part of the system life cycle processes.

The assurance level of a software component is based upon the contribution of software to potential failure conditions as determined by the system safety assessment process by establishing how an error in a software component relates to the system failure condition(s) and the severity of that failure condition(s). The assurance level establishes the rigor necessary to demonstrate compliance with this document.

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

Only partitioned software components (see 2.4.1) can be assigned individual assurance levels by the system safety assessment process. If partitioning between software components cannot be demonstrated, the software components should be viewed as a single component when assigning assurance levels, that is, all components are assigned the assurance level associated with the most severe failure condition to which the software can contribute.

The applicant should establish the system safety assessment process to be used based on approval authority guidance. The assurance level for each software component of the system should then be assigned based on this process and agreed with the approval authorities.

### 2.3.1 Relationship between Software Errors and Failure Conditions

Figure 2-2 shows a sequence of events in which a software error leads to the failure condition at system level. A software error may be latent and, thus, not immediately produce a failure. This model is intentionally a simple, linear representation. In actual operation the sequence of events that leads from a software error to a failure condition may be complex and not easily represented by a sequence of events as shown in Figure 2-2.

It is important to realize that the likelihood that the software contains an error cannot be quantified in the same way as for random hardware failures.

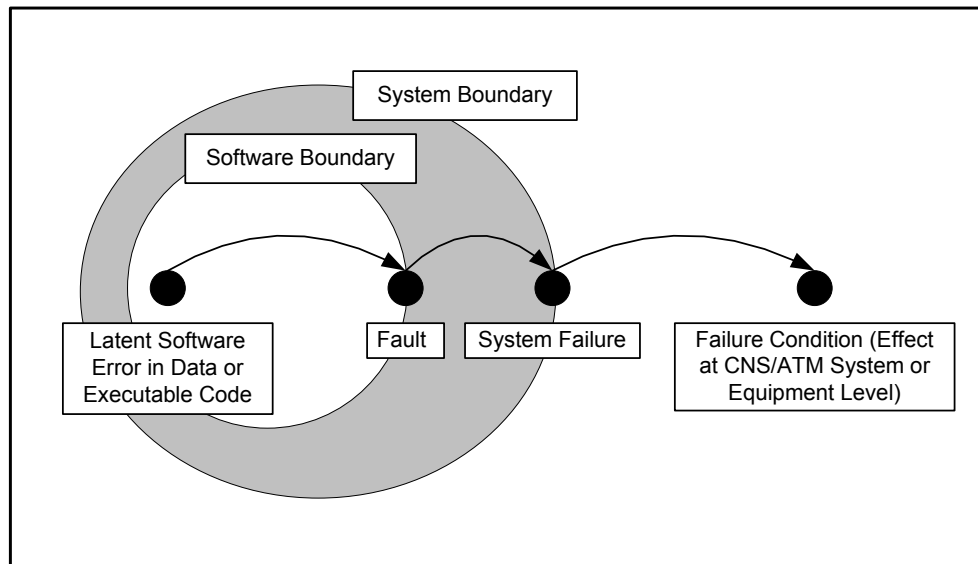


**FIGURE 2-2: SEQUENCE OF EVENTS FOR SOFTWARE ERROR LEADING TO A FAILURE CONDITION**

Architectural considerations (see 2.4) and/or system external factors may also be considered as part of the system safety assessment process when assigning the assurance level to each software component.

### 2.3.2 Assurance Level Definitions

In broad terms, the consequences of a software error related failure condition at CNS/ATM system of equipment level falls in to two categories:

- Those failure conditions with a potential impact upon the safety of the airspace (for example, the loss of separation).
- Those failure conditions which have no likely consequential impact on the airborne domain or the flying public, but may, for example, contribute to the exposure of ground personnel to safety-related risks and hazards (for example, through the anomalous behavior of the software controlling radiation cut-outs for ground-based radar systems under maintenance).

For a complete definition of relevant failure condition categories, the applicant should refer to applicable regulations and guidance material issued by or recognized by the relevant approval authority. The example failure condition categories listed in Table 2-1 are valid for large transport aircraft based on established advisory material for the system safety assessment process. They are only included as examples to assist in the use of this document.

**TABLE 2-1: EXAMPLE FAILURE CONDITION CATEGORY DESCRIPTIONS**

| Category         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Catastrophic     | Failure conditions, which would result in multiple fatalities, usually with the loss of the aircraft.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Hazardous        | Failure conditions which would reduce the capability of the aircraft or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be the following: <ul style="list-style-type: none"> <li>• A large reduction in safety margins or functional capabilities;</li> <li>• Physical distress or higher workload such that the flight crew cannot be relied upon to perform their tasks accurately or completely, or</li> <li>• Serious or fatal injury to a relatively small number of the occupants other than the flight crew.</li> </ul> |
| 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 the flight crew, or physical distress to passengers or cabin crew, possibly including injuries.                                                                                                                   |
| Minor            | Failure conditions which would not significantly reduce aircraft safety, and which 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 physical discomfort to passengers or cabin crew.                                                                                                                                                                                 |
| No Safety Effect | Failure conditions that would have no effect on safety; for example, failure conditions that would not affect the operational capability of the aircraft or increase crew workload.                                                                                                                                                                                                                                                                                                                                                                                                    |

### 2.3.3 Assurance Level Definitions

This document recognizes six assurance levels (AL1 to AL6). For the example failure condition categories listed in section 2.3.2, the relationships between these assurance levels and failure conditions are:

- Level AL1: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of a CNS/ATM system function resulting in a catastrophic failure condition for the aircraft.
- Level AL2: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of a CNS/ATM system function resulting in a hazardous failure condition for the aircraft.
- Level AL3: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of a CNS/ATM system function resulting in a major failure condition for the aircraft.
- Level AL4 is not associated with any failure condition category. Annex A summarizes those objectives, activities and other attributes to be applied to software assigned AL4.
- Level AL5: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of a CNS/ATM system function resulting in a minor failure condition for the aircraft.

- f. Level AL6: Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of a CNS/ATM system function with no effect on aircraft operational capability or pilot workload. If a software component is determined to be AL6 and this is confirmed by the approval authority, no further guidance contained in this document applies.

The applicant should always consider the appropriate approval guidance and system development considerations for allocation of the assurance level. Nothing precludes the assignment of assurance levels to failure condition categories other than those shown in Table 2-1.

Table 2-2 associates the CNS/ATM assurance levels with the airborne software levels. The definitions of the levels associated in Table 2-2 are not identical, but the assurance is equivalent.

**TABLE 2-2: CNS/ATM TO AIRBORNE LEVEL ASSOCIATION**

| ED-109A Assurance Level | ED-12C Software Level |
|-------------------------|-----------------------|
| AL1                     | A                     |
| AL2                     | B                     |
| AL3                     | C                     |
| AL4                     | No Equivalent         |
| AL5                     | D                     |
| AL6                     | E                     |

#### 2.3.4 Assurance Level Determination

The system safety assessment process determines the assurance level(s) appropriate to the software components of a particular system based upon the failure condition which may result from anomalous behavior of the software. The impact of, both loss of function and malfunction should be analyzed. External factors such as adverse environmental conditions as well as architectural strategies (as described in section 2.4) may be considered when determining the assurance level.

**NOTE:** 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 higher assurance level. It may be desirable to develop the software to an assurance 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 assurance level application may be difficult.

If the anomalous behavior of a software component contributes to more than one failure condition, then the software component should be assigned the assurance level associated with the most severe failure condition to which the software can contribute, including combined failure conditions.

#### 2.4 ARCHITECTURAL CONSIDERATIONS

This section provides information on several architectural strategies that may limit the impact of failures, or detect failures and provide acceptable system responses to contain them. These architectural techniques are typically identified during system design and should not be interpreted as the preferred or required solutions.

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 assurance level associated with the failure condition of that system function.

In implementations where anomalous behavior of two or more partitioned software components is needed in order to cause the failure condition, this may be taken into consideration by the system safety assessment process when assigning the assurance level for these software components.

The system safety assessment process needs to establish that sufficient independence exist between software components with respect to both function, that is, high-level requirements and design (for example, common design elements, languages, and tools).

If partitioning and independence between software components cannot be demonstrated, the software components should be viewed as a single software component when assigning assurance levels (that is, all components are assigned the assurance level associated with the most severe failure condition to which the software can contribute).

#### **2.4.1 Partitioning**

Partitioning is a technique for providing isolation between software components to contain and/or isolate faults and potentially reduce the effort of the software verification process. Partitioning between software components may be achieved by allocating unique hardware resources to each component (that is, only one software component is executed on each hardware platform in a system). Alternatively, partitioning provisions may be made to allow multiple software components to run on the same hardware platform. Regardless of the method, the following should be ensured for partitioned software components:

- a. A partitioned software component should not be allowed to contaminate another partitioned software component's code, input/output (I/O), or data storage areas.
- b. A partitioned software component should be allowed to consume shared processor resources only during its scheduled period of execution.
- c. Failures of hardware unique to a partitioned software component should not cause adverse effects on other partitioned software components.
- d. Any software providing partitioning should have the same or higher assurance level as the highest level assigned to any of the partitioned software components.
- e. Any hardware providing partitioning should be assessed by the system safety assessment process to ensure that it does not adversely affect safety.

The software life cycle processes should address the partitioning design considerations. These include 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.

#### **2.4.2 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, multi-version independent 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 that 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 using comparator differences greater than threshold limits. Dissimilar software versions are usually used, therefore, as a means of providing additional protection after the software verification process objectives for the assurance 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 section 12.3.2.

### 2.4.3 Safety Monitoring

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

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

- a. Assurance level: Safety monitoring software is assigned the assurance 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 failure condition.

## 2.5 ADDITIONAL SYSTEM CONSIDERATIONS

### 2.5.1 System Communication

As different hardware and software components communicate with each other, faults in one component have the potential to propagate to other components. Safety analyses should consider data paths through a system to ensure a function of one assurance level does not corrupt data or associated functions of a higher assurance level. CNS/ATM systems development should specifically address fault propagation and data paths.

### 2.5.2 Security

Security requirements should be addressed for CNS/ATM systems. Potential conflicts between security and safety requirements should be resolved at the system level. The resultant requirements to be allocated to software should be defined.

### 2.5.3 Adaptability

The safety considerations of system adaptation should be addressed for CNS/ATM systems.

#### 2.5.4 Cutover (Hot Swapping)

CNS/ATM systems are sometimes required to run 24 hours per day. Some CNS/ATM systems need provisions for replacing components or software while the system is operational. This approach is commonly referred to as “cutover” or “hot swapping.” Cutover considerations include, but are not limited to:

- a. Mechanisms for installing and hot swapping software should ensure that appropriate availability and integrity levels are maintained.
- b. Assurance should be provided such that the safety objectives defined in a safety assessment process have not been compromised during or after cutover.
- c. After a swap, the ability to revert to the previous configuration should be provided.
- d. Compatibility with other system components and the appropriate software version should be maintained.

#### 2.5.5 Post-Development Life Cycle

Users of CNS/ATM systems should address long-term maintenance, spares, retirement, training, etc. While the guidance contained in this document primarily addresses the development of the software for CNS/ATM systems, this does not negate the need to address post-development maintenance and revision activities.

### 2.6 SOFTWARE CONSIDERATIONS IN SYSTEM LIFE CYCLE PROCESSES

This section provides an overview of those software-related issues (not necessarily mutually exclusive) that should be considered, as appropriate, by the system life cycle processes:

- a. Adaptation data items.
- b. User-modifiable software.
- c. Commercial-Off-The-Shelf (COTS) software.
- d. Option-selectable software.
- e. Field-loadable Software.
- f. Software considerations in system verification.

#### 2.6.1 Adaptation Data Items

Software consists of Executable Object Code and/or data, and can comprise one or more configuration items. A data set or an adaptation data set that influences the behavior of the software without modifying the Executable Object Code and is managed as a separate configuration item is called an adaptation data item. That is, when discussing adaptation data items and Executable Object Code, it is implied that the adaptation data items are not part of the Executable Object Code.

An adaptation data item comprises a structure of individual elements where each element can be assigned a single value. Each element has attributes such as type, range, or set of allowed values.

Examples of adaptation data items include configuration tables and databases but not aeronautical data as they are beyond the scope of this document. Adaptation data items may contain data that can:

- a. Influence paths executed through the Executable Object Code.
- b. Activate or deactivate software components and functions.
- c. Adapt the software computations to the system configuration.
- d. Be used as computational data.
- e. Establish time and memory partitioning allotments.

- f. Provide initial values to the software component.
- g. Define physical parameters of equipment such as: data rates, transmission frequency band, router configuration files, port addresses, etc.
- h. Define site specific parameters such as:
  - Geographical data: Latitude, longitude, and elevation, or mask angles corresponding to the location of a fixed asset or reference, for example, radar site, navigation aid, and waypoint.
  - Environmental data: Dynamic data associated with the status of resources, weather information, and flow constraint information.
  - Airspace data: Three-dimensional (3-D) volume definitions specifying the boundary of airspace components. Examples of airspace data include enroute sector boundaries, enroute facility boundaries and approach control facility boundaries. Additional attributes to support specific air traffic control functionality can be associated with airspace data, for example, flight data processing coordination and control.

Depending on how the adaptation data item is to be used in the CNS/ATM system, the following should be addressed:

- User-modifiable software guidance.
- Option-selectable software guidance. In cases where the adaptation data item activates or deactivates functions, the guidance for deactivated code should be addressed as well.
- Field-loadable software guidance. Of particular concern is detection of corrupted adaptation data items, as well as incompatibility between the Executable Object Code and adaptation data items.

An adaptation data item should be assigned the same assurance level as the software component using it.

## 2.6.2 User-Modifiable Software

A user-modifiable component is that part of the software that may be changed by the user within the modification constraints without approval authority review, if the system requirements provide for user modification. A non-modifiable component is that which is not intended to be changed by the user. 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 section 5.2.3. A change that affects the non-modifiable software, its protection, or the modifiable software boundaries is a software modification and is discussed in section 12.1.1.

Guidance for user-modifiable software includes:

- a. The user-modifiable software should not adversely affect safety, operational capabilities, CNS/ATM system user workload, or any non-modifiable software components or any software protection mechanism used. Unless this can be established, the software may not be classified as user-modifiable. The safety impact of displaying information based on user-modifiable software should also be considered.
- b. When the system requirements provide for user modification, then users may modify software within the modification constraints without approval authority review.
- c. The system requirements should specify the mechanisms that prevent the user modification from affecting system safety whether or not they are correctly implemented. The software that provides the protection for user modification should be at the same assurance level as the function it is protecting from errors in the modifiable component.

- d. If the system requirements do not include provision for user modification, then the software should not be modified by the user unless compliance with this document is demonstrated for the modification.
- e. 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.
- f. The applicant should provide the necessary information to enable the user to manage the software in such a way that the safety of the CNS/ATM system is not compromised.

### **2.6.3 Commercial-Off-The-Shelf Software**

COTS software included in CNS/ATM systems or equipment should satisfy the objectives of this document. Additional information may be found in section 12.4.

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 guidance in section 12.1.4, Upgrading a Development Baseline, and section 12.3.4, Product Service Experience, may be relevant in this instance.

### **2.6.4 Option-Selectable Software**

Some CNS/ATM systems and equipment may include optional functions that 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 sections 4.2h, 5.2.4, and 6.4.4.3d(2) for guidance on deactivated code.

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

### **2.6.5 Field-Loadable Software**

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

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.
- CNS/ATM and 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 assurance level as the highest assurance level associated with the function that uses the software load.
- b. If a system recovers to a default mode or safe state upon detection of a corrupted or inappropriate software load, then each partitioned component of the system should have safety-related requirements specified for recovery to and operation in this mode. Interfacing systems may also need to be reviewed for proper operation with the default mode.
- c. The software loading function, including support systems and procedures, should include a means to detect incorrect software and/or hardware and/or CNS/ATM combinations and should provide protection appropriate to the failure condition of the function. If the software consists of multiple configuration items their compatibility should be ensured.
- d. If software is part of an CNS/ATM display mechanism that is the means for ensuring that the CNS/ATM system conforms to an approved configuration, then that software should either be developed to the highest assurance 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.

#### **2.6.6 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 and may be able to satisfy some system verification process objectives. Software design details that relate to the system functionality need to be made available to aid system verification.

### **2.7 SYSTEM CONSIDERATIONS IN SOFTWARE LIFE CYCLE PROCESSES**

Credit may be taken from system life cycle processes for the satisfaction, or partial satisfaction, of the software objectives as defined in this document. In such cases, the system activities for which credit is being sought should be shown to meet the applicable objectives of this document with evidence of the completion of planned activities and their outputs identified as part of the software life cycle data.

## CHAPTER 3

### SOFTWARE LIFE CYCLE

This section discusses the software life cycle processes, software life cycle definition, and transition criteria between software life cycle processes. This document does not prescribe preferred software life cycles and 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:

- a. 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.
- b. 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.
- c. The integral processes that ensure the correctness of control of, and confidence in 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 approval liaison process. It is important to understand that the integral processes are performed concurrently with the software planning and 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 software, 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 an approved system. 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.

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.

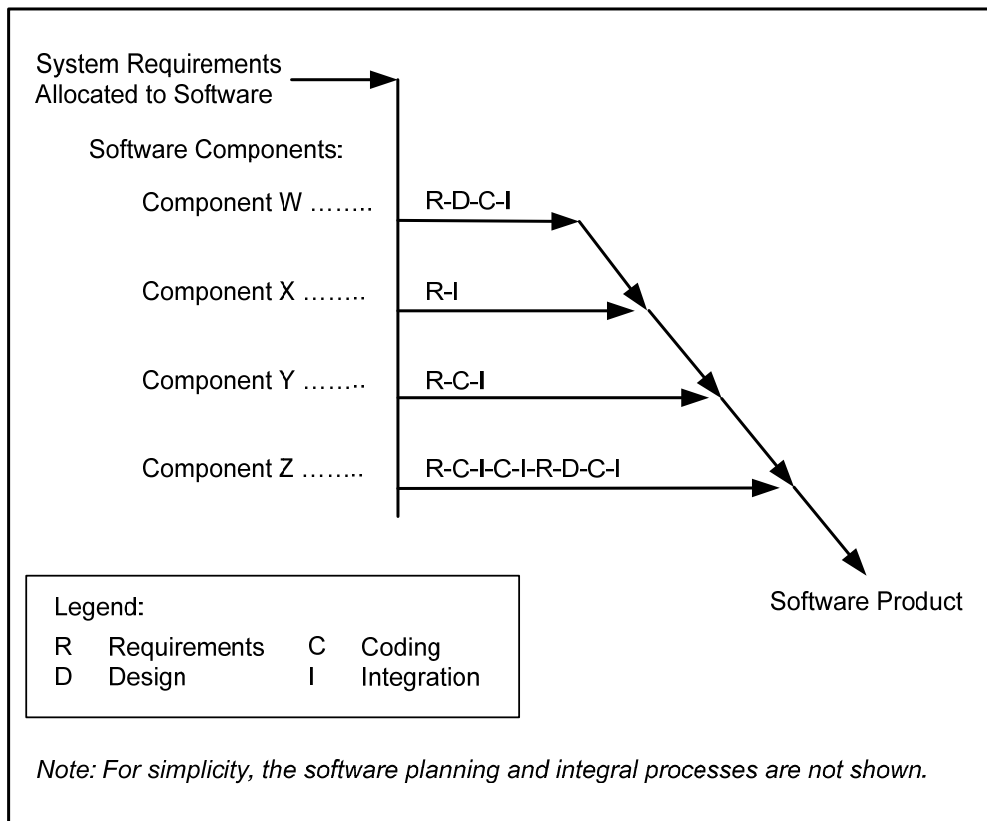


**FIGURE 3-1: EXAMPLE OF A SOFTWARE PROJECT USING FOUR DIFFERENT DEVELOPMENT SEQUENCES**

### 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. The definition of feedback includes how information is recognized, controlled, and resolved by the receiving process. An example of a feedback 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 assurance level. Examples of transition criteria that may be chosen are: 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.

If a process acts on partial inputs, the inputs to the process should be examined to ensure that they meet transition criteria. Also, 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.

## CHAPTER 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 assurance level.

#### 4.1 SOFTWARE PLANNING PROCESS OBJECTIVES

The purpose of the software planning process is to define the means of producing software that will satisfy its requirements and provide the level of confidence that is consistent with the assurance level. 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 assurance level(s) are defined (see 4.2).
- b. The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria are determined (see 3).
- c. The software life cycle environment, including the methods and tools to be used for the activities of each software life cycle process has been selected and defined (see 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 (see 4.5).
- f. Software plans that comply with sections 4.3 and 11 have been produced.
- g. Development and revision of the software plans are coordinated (see 4.3).

#### 4.2 SOFTWARE PLANNING PROCESS ACTIVITIES

Effective planning is a determining factor in producing software that satisfies the guidance of this document. Activities for the software planning process include:

- a. The software plans should be developed that provide direction to the personnel performing the software life cycle processes. Also see section 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 aid error prevention and provide defect detection 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.
- e. The means should be specified 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 dissimilarity 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 (see 4.6).

- h. If deactivated code is planned, the software planning process should describe how the deactivation mechanism and deactivated code will be defined and verified to satisfy system safety objectives.
- i. If user-modifiable software is planned, related processes, tools, environment, and data items substantiating the design (see 5.2.3) should be specified in the software plans and standards.
- j. When adaptation data items are planned, the following should be addressed:
  - 1. The way that adaptation data items are used.
  - 2. The assurance level of the adaptation data items.
  - 3. The processes to develop, verify, and modify adaptation data items, and any associated tool qualification.
  - 4. Software load control and compatibility.
- k. The software planning process should address any additional considerations that are applicable.
- l. If software development activities will be performed by a supplier, planning should address supplier oversight.

Other software life cycle processes may begin before completion of the software planning process if transition criteria for the specific process activity are satisfied.

### 4.3

#### SOFTWARE PLANS

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

Activities for the software plans include:

- a. The software plans should comply with this document.
- b. The software plans should define the transition criteria for 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 approved system. Such changes may be as a result of feedback from other processes and may cause a change to the software plans.

#### 4.4 SOFTWARE LIFE CYCLE ENVIRONMENT PLANNING

Planning for the software life cycle environment defines the methods, tools, procedures, programming languages, and hardware that will be used to develop, verify, control, and produce the software life cycle data (see 11) and software product. Examples of how the software environment chosen can have a beneficial effect on the 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:

- a. The methods and notations used in the software requirements process and software design process.
- b. The programming language(s) and methods used in the software coding process.
- c. The software development environment tools.
- d. The software verification and software configuration management tools.
- e. The need for tool qualification (see 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 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. Activities for the selection of software development environment methods and tools include:

- a. During the software planning process, the software development environment should be chosen to reduce its potential risk to the software being developed.
- b. The use of tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together. This selection includes the assessment of the need for tool qualification.
- c. The software verification process activities or software development standards, which include consideration of the assurance level, should be defined to reduce potential software development environment-related errors.
- d. If approval credit is sought for use of the tools in combination, the sequence of operation of the tools should be specified in the appropriate plan.
- e. If optional features of software tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan. This is especially important for compilers and autocode generators.

- f. Known tool problems and limitations should be assessed and those issues which can adversely affect CNS/ATM software should be addressed.

#### 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 needs 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. Activities include:

- a. Some compilers have features intended to optimize performance of the object code. If the test cases give coverage consistent with the assurance level, the correctness of the optimization need not be verified. Otherwise, the impact of these features on structural coverage analysis should be determined. Additional information can be found in section 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 (see 6.4.4.2.b). The software planning process should provide a means to detect this object code and to ensure verification coverage, and should 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 that is consistent with sections 6 and 12.1.3.

**NOTE:** Although the compiler is considered acceptable once all of the verification objectives are satisfied, the compiler is only considered acceptable for that product and not necessarily for other products.

#### 4.4.3 Software Test Environment

Software test environment planning defines 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. Activities include:

- a. The emulator or simulator may need to be qualified as described in section 12.2.
- b. The differences between the target computer and the emulator or simulator, and the effects of these differences on the ability to detect errors and verify functionality, should be considered. Detection of those errors should be provided by the software verification process and specified in the Software Verification Plan.

#### 4.5 SOFTWARE DEVELOPMENT STANDARDS

Software development standards 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. Activities for development of the software standards include:

- a. The software development standards should comply with 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.
- d. Robustness should be considered in the software development standards.

**NOTE 1:** In developing standards, consideration can be given to previous experience. Constraints and rules on development, design, and coding methods can be included to control complexity. Defensive programming practices may be considered to improve robustness.

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

#### 4.6

#### REVIEW OF THE SOFTWARE PLANNING PROCESS

Reviews of the software planning process are conducted to ensure that the software plans and software development standards comply with the guidance of this document and means are provided to execute them. Activities include:

- a. Methods should be chosen that enable the objectives of this document to be satisfied.
- b. Software life cycle processes should be able to be applied consistently.
- c. Each process should produce evidence that its outputs can be traced to its 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 should be consistent and comply with section 11.

## CHAPTER 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 (see 4) and the Software Development Plan (see 11.2). Table A-2 of Annex A is a summary of the objectives and outputs of the software development processes by assurance 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 guidance for low-level requirements also apply.

**NOTE:** The applicant may be required to justify software development processes that produce a single level of requirements.

The development of 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. Some examples of requirements that might be determined to be derived requirements are:

- The need for interrupt handling software to be developed for the chosen target computer.
- The specification of a periodic monitor's iteration rate when not specified by the system requirements allocated to software.
- The addition of scaling limits when using fixed point arithmetic.

High-level requirements may include derived requirements, and low-level requirements may include derived requirements. In order to determine the effects of derived requirements on the system safety assessment and system requirements, all derived requirements should be made available to the system processes including the system safety assessment process.

## 5.1 SOFTWARE REQUIREMENTS PROCESS

The software requirements process uses the outputs of the system life cycle processes to develop the 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 defined and provided to the system processes, including the system safety assessment process.

### 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 high-level requirements.

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

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

- 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, be verifiable, and be 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. Derived high-level requirements and the reason for their existence should be defined.
- i. Derived high-level requirements should be provided to the system processes, including the system safety assessment process.
- j. If adaptation data items are planned, the high-level requirements should describe how any adaptation data item is used by the software. The high-level requirements should also specify their structure, the attributes for each of their data elements, and, when applicable, the value of each element. The values of the adaptation data item elements should be consistent with the structure of the adaptation data item and the attributes of its data elements.

## 5.2 SOFTWARE DESIGN PROCESS

The 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 defined and provided to the system processes, including the system safety assessment process.

### 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 (see 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. Activities for this process include:

- 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 low-level requirements and the reason for their existence 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 assurance level assignment for some components of the software. In such cases, additional data should be defined as derived requirements and provided to the system processes, including the system safety assessment process.
- d. Interfaces between software components, in the form of data flow and control flow, should be defined to be consistent between the components.
- e. Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks, and cross-channel comparisons.
- f. Responses to failure conditions should be consistent with the safety-related requirements.
- g. Inadequate or incorrect inputs detected during the software design process should be provided to the system life cycle processes, 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 system safety objectives. While no objective guidance 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-related requirements are satisfied.

### 5.2.3 Designing for User-Modifiable Software

User-modifiable software is designed to be modified 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 maintenance functions. Software of any level can include a modifiable component.

The activities for user-modifiable software include:

- 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. If the protection is provided by software, it should be designed and verified at the same assurance level as the non-modifiable software. If the protection is provided by a tool, the tool should be categorized and qualified as defined in section 12.2.
- b. The means provided to change the modifiable component should be shown to be the only means by which the modifiable component can be changed.

### 5.2.4 Designing for Deactivated Code

Systems 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, such as unselected functionality or unused library functions, or data that is not used. Deactivated code differs from dead code. The activities for deactivated code include:

- a. A mechanism should be designed and implemented to assure that deactivated functions or components have no adverse impact on active functions or components.
- b. Evidence should be available that the deactivated code is disabled for the environments where its use is not intended. Unintended execution of deactivated code due to abnormal system conditions is the same as unintended execution of activated code.
- c. The development of deactivated code, like the development of the active code, should comply with the objectives of this document.

## 5.3 SOFTWARE CODING PROCESS

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

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

### 5.3.1 Software Coding Process Objectives

The objective of the software coding process is:

- a. Source Code is developed from the 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, 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 output of this process is Source Code (see 11.11).

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

- 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. Inadequate or incorrect inputs detected during the software coding process should be provided to the software requirements process, software design process, and/or software planning process as feedback for clarification or correction.
- d. Use of autocode generators should conform to the constraints defined in the planning process.

## **5.4 INTEGRATION PROCESS**

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

### **5.4.1 Integration Process Objectives**

The objective of the integration process is:

- a. The Executable Object Code and its associated Adaptation Data Item Files, if any, are produced and loaded into the target hardware for hardware/software integration.

### **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 from the software coding process.

The outputs of the integration process are the object code; Executable Object Code (see section 11.12); Adaptation Data Item File (see 11.22); and the compiling, linking, and loading data (see 11.11). The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include:

- a. The object code and Executable Object Code should be generated from the Source Code and compiling, linking, and loading data. Any Adaptation Data Item File should be generated.
- b. Software integration should be performed on a host computer, a target computer emulator, or the target computer.
- c. The software should be loaded into the target computer for hardware/software integration.
- d. 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.
- e. Patches should not be used in software submitted for use in an approved system to implement changes in requirements or architecture, or changes found necessary as a result of the 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.

- f. When a patch is used, these should be available:
  - 1. Confirmation that the software configuration management process can effectively track the patch.
  - 2. An analysis to provide evidence that the patched software satisfies all the applicable objectives.
  - 3. Justification in the Software Accomplishment Summary for use of a patch.

## 5.5 SOFTWARE DEVELOPMENT PROCESS TRACEABILITY

Software development process traceability activities include:

- a. Trace Data, showing the bi-directional association between system requirements allocated to software and high-level requirements, is developed. The purpose of this Trace Data is to:
  - 1. Enable verification of the complete implementation of the system requirements allocated to software.
  - 2. Give visibility to those derived high-level requirements that are not directly traceable to system requirements.
- b. Trace Data, showing the bi-directional association between the high-level requirements and low-level requirements is developed. The purpose of this Trace Data is to:
  - 1. Enable verification of the complete implementation of the high-level requirements.
  - 2. Give visibility to those derived low-level requirements that are not directly traceable to high-level requirements and to the architectural design decisions made during the software design process.
- c. Trace Data, showing the bi-directional association between low-level requirements and Source Code, is developed. The purpose of this Trace Data is to:
  - 1. Enable verification that no Source Code implements an undocumented function.
  - 2. Enable verification of the complete implementation of the low-level requirements.

## CHAPTER 6

### SOFTWARE VERIFICATION PROCESS

This section discusses the objectives and activities of the software verification process. Verification is a technical assessment of the outputs of the software planning process, software development processes, and the software verification process. The software verification process is applied as defined by the software planning process (see 4) and the Software Verification Plan (see 11.3). See 4.6 for the verification of the outputs of the planning process.

Verification is not simply testing. Testing, in general, cannot show the absence of errors. As a result, the following sections use the term "verify" instead of "test" to discuss the software verification process activities, which are typically a combination of reviews, analyses, and tests.

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

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

- Verification of Source Code.
- 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 PURPOSE OF SOFTWARE VERIFICATION

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 software verification process verifies that:

- a. The system requirements allocated to software have been developed into 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 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,(that is, intended function) and provides confidence in the absence of unintended functionality.

- e. The Executable Object Code is robust with respect to the software requirements such that it can respond correctly to abnormal inputs and conditions.
- f. The means used to perform this verification are technically correct and complete for the assurance level.

## 6.2

### OVERVIEW OF 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 and procedures 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, software architecture, Trace Data, Source Code, Executable Object Code, and the Software Verification Plan.

The outputs of the software verification process are recorded in the Software Verification Cases and Procedures (see 11.13), the Software Verification Results (see 11.14), and the associated Trace Data (see 11.21).

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.

Software verification considerations include:

- a. If the code tested is not identical to the CNS/ATM software, those differences should be specified and justified.
- b. 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.
- c. Deficiencies and errors discovered during the software verification process should be reported to other software life cycle development processes for clarification and correction as applicable.
- d. Reverification should be conducted following corrective actions and/or changes that could impact the previously verified functionality. Reverification should ensure that the modification has been correctly implemented.
- e. Verification independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified. A tool may be used to achieve equivalence to the human verification activity. For independence, the person who created a set of low-level requirements-based test cases should not be the same person who developed the associated Source Code from those low-level requirements.

## 6.3 SOFTWARE REVIEWS AND ANALYSES

Reviews and analyses are applied to the outputs of the software development processes. 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 system or equipment.

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

### 6.3.1 Reviews and Analyses of High-Level Requirements

These review and analysis activities detect and report requirements errors that may have been introduced during the software requirements process. These review and analysis activities 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 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 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 Low-Level Requirements

These review and analysis activities detect and report requirements errors that may have been introduced during the software design process. These review and analysis activities confirm that the low-level requirements satisfy these objectives:

- a. Compliance with high-level requirements: The objective is to ensure that the low-level requirements satisfy the 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 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.
- 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 Software Architecture

These review and analysis activities detect and report errors that may have been introduced during the development of the software architecture. These review and analysis activities 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. If the interface is to a component of a lower assurance level, it should also be confirmed that the higher assurance level component has appropriate protection mechanisms in place to protect itself from potential erroneous inputs from the lower assurance level component.
- 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, for example, deviations to complexity restriction and design construct rules.
- f. Partitioning integrity: The objective is to ensure that partitioning breaches are prevented.

### 6.3.4

#### Reviews and Analyses of Source Code

These review and analysis activities detect and report errors that may have been introduced during the software coding process. Primary concerns include correctness of the code with respect to the software requirements, the software architecture, and conformance to the Software Code Standards. These review and analysis activities are usually confined to the Source Code and confirm that the Source Code satisfies these objectives:

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