



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

MODEL-BASED DEVELOPMENT AND VERIFICATION SUPPLEMENT TO ED-12C AND ED-109A

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-218

January 2012

MODEL-BASED DEVELOPMENT AND VERIFICATION SUPPLEMENT TO ED-12C AND ED-109A

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-218

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

Telephone: 33 1 40 92 79 30

Fax: 33 1 46 55 62 65

Email: eurocae@eurocae.net

Website: www.eurocae.net

TABLE OF CONTENTS

MB.1.0	INTRODUCTION	1
MB.1.1	Purpose.....	1
MB.1.2	Scope.....	1
MB.1.3	Relationship to Other Documents.....	1
MB.1.4	How to Use This Document	2
MB.1.5	Document Overview	2
MB.1.6	Characteristics of Model-Based Development and Verification	2
	MB.1.6.1 Requirements From Which the Model is Developed.....	3
	MB.1.6.2 Specification Models and Design Models	3
	MB.1.6.3 Examples of Model Usage	3
MB.2.0	SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT	5
MB.2.1	System Requirements Allocation to Software	5
MB.2.2	Information Flow Between System and Software Life Cycle Processes.....	5
	MB.2.2.1 Information Flow from System Processes to Software Processes	7
	MB.2.2.2 Information Flow from Software Processes to System Processes	7
	MB.2.2.3 Information Flow between Software Processes and Hardware Processes ..	8
MB.2.3	System Safety Assessment Process and Software Level.....	8
MB.2.4	Architectural Considerations.....	8
MB.2.5	Software Considerations in System Life Cycle Processes.....	8
	MB.2.5.1 Parameter Data Items	8
	MB.2.5.2 User-Modifiable Software	8
	MB.2.5.3 Commercial-Off-The-Shelf Software	9
	MB.2.5.4 Option-Selectable Software.....	9
	MB.2.5.5 Field-Loadable Software	9
	MB.2.5.6 Software Considerations in System Verification	9
MB.2.6	System Considerations in Software Life Cycle Processes.....	9
MB.3.0	SOFTWARE LIFE CYCLE.....	10
MB.3.1	Software Life Cycle Processes.....	10
MB.3.2	Software Life Cycle Definition.....	10
MB.3.3	Transition Criteria Between Processes	10
MB.4.0	SOFTWARE PLANNING PROCESS	11
MB.4.1	Software Planning Process Objectives.....	11
MB.4.2	Software Planning Process Activities	12
MB.4.3	Software Plans.....	13
MB.4.4	Software Life Cycle Environment Planning	13
	MB.4.4.1 Software Development Environment.....	14
	MB.4.4.2 Language and Compiler Considerations.....	14
	MB.4.4.3 Software Test Environment	14
	MB.4.4.4 Simulation Environment.....	15
MB.4.5	Software Development Standards.....	15
MB.4.6	Review of the Software Planning Process.....	15
MB.5.0	SOFTWARE DEVELOPMENT PROCESSES.....	16
MB.5.1	Software Requirements Process	17
	MB.5.1.1 Software Requirements Process Objectives.....	17
	MB.5.1.2 Software Requirements Process Activities	17
MB.5.2	Software Design Process	18
	MB.5.2.1 Software Design Process Objectives	18

MB.5.2.2	Software Design Process Activities	18
MB.5.2.3	Designing for User-Modifiable Software	19
MB.5.2.4	Designing for Deactivated Code	20
MB.5.3	Software Coding Process	20
MB.5.4	Integration Process	20
MB.5.4.1	Integration Process Objectives	20
MB.5.4.2	Integration Process Activities	20
MB.5.5	Software Development Process Traceability	20
MB.6.0	SOFTWARE VERIFICATION PROCESS	21
MB.6.1	Purpose of Software Verification	21
MB.6.2	Overview of Software Verification Process Activities	22
MB.6.3	Software Reviews and Analyses	23
MB.6.3.1	Reviews and Analyses of High-Level Requirements	23
MB.6.3.2	Reviews and Analyses of Low-Level Requirements	24
MB.6.3.3	Reviews and Analyses of Software Architecture	24
MB.6.3.4	Reviews and Analyses of Source Code	25
MB.6.3.5	Reviews and Analyses of the Outputs of the Integration Process	25
MB.6.4	Software Testing	25
MB.6.5	Software Verification Process Traceability	25
MB.6.6	Verification of Parameter Data Items	26
MB.6.7	Model Coverage Analysis for Design Models	26
MB.6.7.1	Model Coverage Analysis Criteria	26
MB.6.7.2	Model Coverage Analysis Resolution	27
MB.6.8	Model Simulation	28
MB.6.8.1	Model Simulation for Verification of the Model	28
MB.6.8.2	Model Simulation for Verification of the Executable Object Code	29
MB.6.8.3	Simulation Cases, Procedures and Results	31
MB.6.8.3.1	Development of Simulation Cases, Procedures and Results	31
MB.6.8.3.2	Reviews and Analyses of Simulation Cases, Procedures and Results	31
MB.7.0	SOFTWARE CONFIGURATION MANAGEMENT PROCESS	32
MB.7.1	Software Configuration Management Process Objectives	32
MB.7.2	Software Configuration Management Process Activities	33
MB.7.2.1	Configuration Identification	33
MB.7.2.2	Baselines and Traceability	33
MB.7.2.3	Problem Reporting, Tracking, and Corrective Action	33
MB.7.2.4	Change Control	33
MB.7.2.5	Change Review	33
MB.7.2.6	Configuration Status Accounting	33
MB.7.2.7	Archive, Retrieval, and Release	34
MB.7.3	Data Control Categories	34
MB.7.4	Software Load Control	34
MB.7.5	Software Life Cycle Environment Control	35
MB.8.0	SOFTWARE QUALITY ASSURANCE PROCESS	36
MB.8.1	Software Quality Assurance Process Objectives	36
MB.8.2	Software Quality Assurance Process Activities	36
MB.8.3	Software Conformity Review	37

MB.9.0	CERTIFICATION LIAISON PROCESS	38
MB.9.1	Means of Compliance and Planning	38
MB.9.2	Compliance Substantiation	38
MB.9.3	Minimum Software Life Cycle Data Submitted to Certification Authority	38
MB.9.4	Software Life Cycle Data Related to Type Design	38
MB.10.0	OVERVIEW OF CERTIFICATION PROCESS	39
MB.10.1	Certification Basis	39
MB.10.2	Software Aspects of Certification	39
MB.10.3	Compliance Determination	39
MB.11.0	SOFTWARE LIFE CYCLE DATA	40
MB.11.1	Plan for Software Aspects of Certification	41
MB.11.2	Software Development Plan	42
MB.11.3	Software Verification Plan	42
MB.11.4	Software Configuration Management Plan	43
MB.11.5	Software Quality Assurance Plan	44
MB.11.6	Software Requirements Standards	44
MB.11.7	Software Design Standards	44
MB.11.8	Software Code Standards	44
MB.11.9	Software Requirements Data	44
MB.11.10	Design Description	45
MB.11.11	Source Code	45
MB.11.12	Executable Object Code	45
MB.11.13	Software Verification Cases and Procedures	45
MB.11.14	Software Verification Results	46
MB.11.15	Software Life Cycle Environment Configuration Index	46
MB.11.16	Software Configuration Index	46
MB.11.17	Problem Reports	47
MB.11.18	Software Configuration Management Records	47
MB.11.19	Software Quality Assurance Records	47
MB.11.20	Software Accomplishment Summary	47
MB.11.21	Trace Data	47
MB.11.22	Parameter Data Item File	47
MB.11.23	Software Model Standards	48
MB.12.0	ADDITIONAL CONSIDERATIONS	49
MB.12.1	Use of Previously Developed Software	49
MB.12.1.1	Modifications of Previously Developed Software	49
MB.12.1.2	Change of Aircraft Installation	49
MB.12.1.3	Change of Application or Development Environment	49
MB.12.1.4	Upgrading a Development Baseline	50
MB.12.1.5	Software Configuration Management Considerations	51
MB.12.1.6	Software Quality Assurance Considerations	51
MB.12.2	Tool Qualification	51
MB.12.2.1	Determining if Tool Qualification is Needed	51
MB.12.2.2	Determining the Tool Qualification Level	52
MB.12.2.3	Tool Qualification Process	52
MB.12.3	Alternative Methods	52
MB.12.3.1	Exhaustive Input Testing	52
MB.12.3.2	Considerations for Multiple-Version Dissimilar Software Verification	52

MB.12.3.3	Software Reliability Models	52
MB.12.3.4	Product Service History	52
MB.12.3.4.1	Relevance of Service History.....	52
MB.12.3.4.2	Sufficiency of Accumulated Service History	52
MB.12.3.4.3	Collection, Reporting, and Analysis of Problem Reports Found During Service History	52
MB.12.3.4.4	Service History Information to be Included in the Plan for Software Aspects of Certification	53
ANNEX MB.A – PROCESS OBJECTIVES AND OUTPUTS BY SOFTWARE LEVEL IN ED-12C ..		54
ANNEX MB.B - ACRONYMS AND GLOSSARY OF TERMS		68
ANNEX MB.C – PROCESS OBJECTIVES AND OUTPUTS BY ASSURANCE LEVEL IN ED-109A70		
APPENDIX MB.A – COMMITTEE MEMBERSHIP		84
APPENDIX MB.B – FREQUENTLY ASKED QUESTIONS AND DISCUSSION PAPERS		96
MB.B.1	FAQ #1: What is the Data Control Category of a model?	96
MB.B.2	FAQ #2: Which verification objectives does model simulation, by itself support?....	96
MB.B.3	FAQ #3: What verification objectives can be met by combining model simulation with other tools or methods?	96
MB.B.4	FAQ #4: If a model is used to represent requirements and an autocode generator is not used, does the supplement apply?	97
MB.B.5	FAQ #5: How should the applicant develop and verify manually written Source Code, that is invoked by the code generated from the Design Model?	97
MB.B.6	FAQ #6: When Source Code is automatically generated from a Design Model, what type of testing should be performed to provide assurance of the correctness of the integration of this Source Code with manually written Source Code?.....	97
MB.B.7	FAQ #7: When using models for verification, should expected results be determined prior to test execution?	98
MB.B.8	FAQ #8: Is a model subject to Tool Qualification?	98
MB.B.9	FAQ #9: What is an example of a model that is considered part of the test environment and what activities are applicable to that model?	99
MB.B.10	FAQ #10: Can a single Software Model Standard be applied to both Specification Models and Design Models?	99
MB.B.11	FAQ #11: May the applicant use the model coverage analysis activity to achieve the structural coverage analysis objectives?	99
MB.B.12	FAQ #12: What are the independence issues when a Design Model is used for both code generation and test generation?	100
MB.B.13	FAQ #13: How does the supplement apply if a Design Model that is used in the software development process is part of the system-level life cycle data, as in example 5 of Table MB.1-1 (that is, the Design Model also contains system requirements allocated to software)?	101
MB.B.14	FAQ #14: How does the supplement apply if the requirements from which the Design Model was developed are part of the system life cycle data, as in example 4 and 5 of Table MB.1-1 (that is, high-level requirements per ED-12C and this supplement are at the system level in the form of system requirements allocated to software)? ..	101
MB.B.15	FAQ #15: Do data files associated with models need to be treated as parameter data items?	102
MB.B.16	FAQ #16: Can simulation support the assessment of test coverage of the low-level requirements contained in a Design Model?	102
MB.B.17	DP #1: Examples of model-based development and the relationship between a Design Model or a Specification Model and DO178C high-level requirements, low-level requirements, and software architecture.....	103
MB.B.18	DP #2: Information on the usage of libraries in a Model-Based Development and Verification processes.....	109

LIST OF FIGURES

FIGURE MB.2-1	INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES.....	6
FIGURE MB.DP1-1	KEY TO READING DP1 DIAGRAMS.....	103
FIGURE MB.DP1-2	EXAMPLE A: SEPARATE MODELS USED TO EXPRESS LLR AND SW ARCHITECTURE.....	104
FIGURE MB.DP1-3	EXAMPLE B: ONE MODEL USED TO EXPRESS HLR, LLR/SW ARCHITECTURE WITH HLR PROVIDED BY SRATS.....	105
FIGURE MB.DP1-4	EXAMPLE C: SEPARATE MODELS USED TO EXPRESS HLR AND LLR/SW ARCHITECTURE.....	106
FIGURE MB.DP1-5	EXAMPLE D: ONE MODEL USED TO EXPRESS HLR AND ONLY HLR..	107
FIGURE MB.DP1-6	EXAMPLE E: ONE MODEL PROVIDED BY SRATS USED TO EXPRESS SYSTEM REQUIREMENTS, HLR, LLR/SW ARCHITECTURE	108

LIST OF TABLES

TABLE MB.1-1	MODEL USAGE EXAMPLES	4
TABLE MB.6-1	MODEL COVERAGE CRITERIA EXAMPLE	27
TABLE MB.7-1	SCM PROCESS ACTIVITIES ASSOCIATED WITH CC1 AND CC2 DATA.	34
TABLE MB.A-1	SOFTWARE PLANNING PROCESS.....	55
TABLE MB.A-2	SOFTWARE DEVELOPMENT PROCESSES.....	56
TABLE MB.A-3	VERIFICATION OF OUTPUTS OF SOFTWARE REQUIREMENTS PROCESS	58
TABLE MB.A-4	VERIFICATION OF OUTPUTS OF SOFTWARE DESIGN PROCESS	59
TABLE MB.A-5	VERIFICATION OF OUTPUTS OF SOFTWARE CODING & INTEGRATION PROCESSES	61
TABLE MB.A-6	TESTING OF OUTPUTS OF INTEGRATION PROCESS	62
TABLE MB.A-7	VERIFICATION OF VERIFICATION PROCESS RESULTS	63
TABLE MB.A-8	SOFTWARE CONFIGURATION MANAGEMENT PROCESS.....	65
TABLE MB.A-9	SOFTWARE QUALITY ASSURANCE PROCESS	66
TABLE MB.A-10	CERTIFICATION LIAISON PROCESS	67
TABLE MB.C-1	SOFTWARE PLANNING PROCESS.....	71
TABLE MB.C-2	SOFTWARE DEVELOPMENT PROCESSES.....	72
TABLE MB.C-3	VERIFICATION OF OUTPUTS OF SOFTWARE REQUIREMENTS PROCESS	74
TABLE MB.C-4	VERIFICATION OF OUTPUTS OF SOFTWARE DESIGN PROCESS	75
TABLE MB.C-5	VERIFICATION OF OUTPUTS OF SOFTWARE CODING & INTEGRATION PROCESSES	77
TABLE MB.C-6	TESTING OF OUTPUTS OF INTEGRATION PROCESS	78
TABLE MB.C-7	VERIFICATION OF VERIFICATION PROCESS RESULTS	79
TABLE MB.C-8	SOFTWARE CONFIGURATION MANAGEMENT PROCESS	81
TABLE MB.C-9	SOFTWARE QUALITY ASSURANCE PROCESS	82
TABLE MB.C-10	SOFTWARE APPROVAL PROCESS	83

MB.1.0

INTRODUCTION

A model is an abstract representation of a set of software aspects of a system that is used to support the software development process or the software verification process. This supplement addresses model(s) that have the following characteristics:

- a. The model is completely described using an explicitly identified modeling notation. The modeling notation may be graphical and/or textual.
- b. The model contains software requirements and/or software architecture definition.
- c. The model is of a form and type that is used for direct analysis or behavioral evaluation as supported by the software development process or the software verification process.

The following are not considered as models within this supplement:

- Figures without syntax/semantics (these may be illustrative).
- Equations related to natural language sentences.

The use of models may bring the benefits and capabilities of:

- Providing unambiguous expression of requirements and architecture.
- Supporting the use of automated code generation.
- Supporting the use of automated test generation.
- Supporting the use of analysis tools for verification of requirements and architecture.
- Supporting the use of simulation for partial verification of requirements, architecture, and/or Executable Object Code.

Since the publication of ED-12B, advances and experience have been gained in model-based development and verification, their application, and supporting tools. As the use of this technology for critical software applications in avionics has increased, there are a number of issues that need to be considered to ensure the safety and integrity goals are met. Clarifying each of these issues will ease the application of model-based development and verification; therefore, this supplement provides guidance for applicants and certification or approval authorities to facilitate the use of this technology.

MB.1.1 PURPOSE

This supplement contains modifications and additions to ED-12C objectives, activities, explanatory text, and software life cycle data that should be addressed when model-based development and verification are used as part of the software life cycle. This includes the artifacts that would be expressed using models and the verification evidence that could be derived from them. Therefore, this supplement also applies to the models developed in the system process that define software requirements or software architecture.

MB.1.2 SCOPE

This supplement discusses the use of model-based development and verification in the software life cycle for software that is produced in accordance with ED-12C. If the applicant is planning to use model-based development and verification, then the applicant should comply with this supplement. This supplement does not provide guidance for models developed and/or used only for the software verification process.

MB.1.3 RELATIONSHIP TO OTHER DOCUMENTS

This document supplements the guidance given in ED-12C. This document may be used in conjunction with other ED-12C supplements.

MB.1.4 HOW TO USE THIS DOCUMENT

This document should be used with and in the same way as ED-12C; however, the following should be noted:

- a. Section MB.1 contains explanatory text to aid the reader in understanding model-based development and verification, and therefore is not to be taken as guidance.
- b. Annex MB.A of this supplement describes how the ED-12C objectives are revised in line with this model-based development and verification guidance.
- c. Guidance from ED-12C should be used for all aspects of the software life cycle, apart from where model-based development and verification are used. All guidance specific to model-based development and verification is contained within this supplement.
- d. This supplement can be applied to ED-109A in the same manner as ED-12C. Applicants complying with ED-109A should be aware that the text within this supplement is based on ED-12C; thus the ED-109A applicant needs to ensure that the proper ED-109A guidance is being addressed. Annex MB.C of this document describes how the ED-109A, Annex A objectives are revised in line with model-based development and verification guidance in this supplement.
- e. In addition to the guidance contained in this supplement, this supplement also provides clarification on the guidance provided in the form of answers to Frequently Asked Questions (FAQs) and Discussion Papers (DPs) (see Appendix MB.B). Neither the FAQs nor the DPs contain new or additional guidance.
- f. Major sections are numbered as MB.X.0 throughout this supplement. It should be noted that references to an entire section are identified as "section MB.X"; whereas, references to the content between section headers MB.X.0 and MB.X.1 are referred to as "section MB.X.0".

MB.1.5 DOCUMENT OVERVIEW

This supplement is intended to be used with ED-12C. Sections MB.2 through MB.12 and Annex MB.A provide additional and modified text to the referenced sections of ED-12C. The document strives to minimize redundancy between ED-12C and the document itself. Each affected section of ED-12C is included in this document. Where the guidance from ED-12C applies unchanged, this is stated in this document. Text taken directly from ED-12C is shown in italics. For those sections from ED-12C included in this supplement that are unchanged, except for the references, the ED-12C text is included with the updated MB reference.

In Annex MB.A, only those tables with changes are included. These tables include new objectives, activities, and changes to referenced text for existing objectives.

MB.1.6 CHARACTERISTICS OF MODEL-BASED DEVELOPMENT AND VERIFICATION

As part of the software development and verification processes, models may represent software requirements and/or architecture, partially or completely.

Models are characterized by the modeling technique that is used and by the type of software life cycle data they will represent. A modeling technique is a combination of a modeling language and a particular manner of using this modeling language. The ability of a model to correctly and completely express a certain type of software life cycle data depends on the modeling technique used. Modeling techniques used should be suitable to the type and to the level of abstraction of the information expressed by the model. Software Model Standards are the means to describe a modeling technique and support the demonstration of how this technique is fit for its purpose.

MB.1.6.1 Requirements from which the Model is Developed

For any type of model within the scope of this supplement, there is a need to identify the requirements from which the model is developed. Requirements from which the model is developed should be external to the model. Those requirements should provide details and constraints to enable model development and verification activities.

MB.1.6.2 Specification Models and Design Models

This supplement introduces two types of models: Specification Models and Design Models.

A Specification Model represents high-level requirements that provide an abstract representation of functional, performance, interface, or safety characteristics of software components. The Specification Model should express these characteristics unambiguously to support an understanding of the software functionality. It should only contain detail that contributes to this understanding and does not prescribe a specific software implementation or architecture except for exceptional cases of justified design constraints. Specification models do not define software design details such as internal data structures, internal data flow or internal control flow. Therefore, a Specification Model may express high-level requirements but neither low-level requirements nor software architecture.

A Design Model prescribes software component internal data structures, data flow, and/or control flow. A Design Model includes low-level requirements and/or architecture. In particular, when a model expresses software design data, regardless of other content, it should be classified as a Design Model. This includes models used to produce code.

A model cannot be classified as both a Specification Model and a Design Model.

For the purpose of this supplement, the term “high-level requirement” refers to either of the following:

- Any requirement contained in a Specification Model.
- Any requirement from which a Design Model is developed.

For the purpose of this supplement, the term “low-level requirement” refers to:

- Any requirement contained in a Design Model.

MB.1.6.3 Examples of Model Usage

Table MB.1-1 reflects some of the current industry practices used for the development of software requirements and/or architecture.

TABLE MB.1-1: MODEL USAGE EXAMPLES

Process that generates the life-cycle data	MB Example 1	MB Example 2	MB Example 3	MB Example 4 (See Note 1)	MB Example 5 (See Note 1)
System Requirement and System Design Processes	Requirements allocated to software	Requirements from which the Model is developed	Requirements from which the Model is developed	Requirements from which the Model is developed	Requirements from which the Model is developed
					Design Model
Software Requirement and Software Design Processes	Requirements from which the Model is developed	Specification Model (See Note 2)	Specification Model	Design Model	
	Design Model	Design Model	Textual description (See Note 3)		
Software Coding Process	Source Code	Source Code	Source Code	Source Code	Source Code

NOTE 1: In MB Example 4 and MB Example 5 it is difficult to separate system and software life cycle data. Regardless of where the model originated, the supplement applies and the objectives of this supplement should be satisfied. Therefore, in these examples the guidance related to low-level requirements should be applied to the Design Model and the guidance related to high-level requirements should be applied to the requirements from which the model is developed. See the explanatory text below [Table MB.A-3](#) of Annex MB.A related to the applicability of objectives in this case.

NOTE 2: In MB Example 2, the Design Model is developed from the requirements contained in the Specification Model.

NOTE 3: In MB Example 3, the textual description refers to low-level requirements (and possibly software architecture); ED-12C guidance is applicable to these.

MB.2.0

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 (for example, SAE ARP4754A).

Discussed in this section are:

- *System requirements allocation to software (see ED-12C section 2.1).*
- *The information flow between the system and software life cycle processes and between the software and hardware life cycle processes (see MB.2.2).*
- *The system safety assessment process, failure conditions, software level definitions, and software level determination (see ED-12C section 2.3).*
- *Architectural considerations (see ED-12C section 2.4).*
- *Software considerations in system life cycle processes (see MB.2.5).*
- *System considerations in software life cycle processes (see ED-12C section 2.6).*

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

MB.2.1 SYSTEM REQUIREMENTS ALLOCATION TO SOFTWARE

Section 2.1 of ED-12C is unchanged.

MB.2.2 INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES

Figure MB.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.

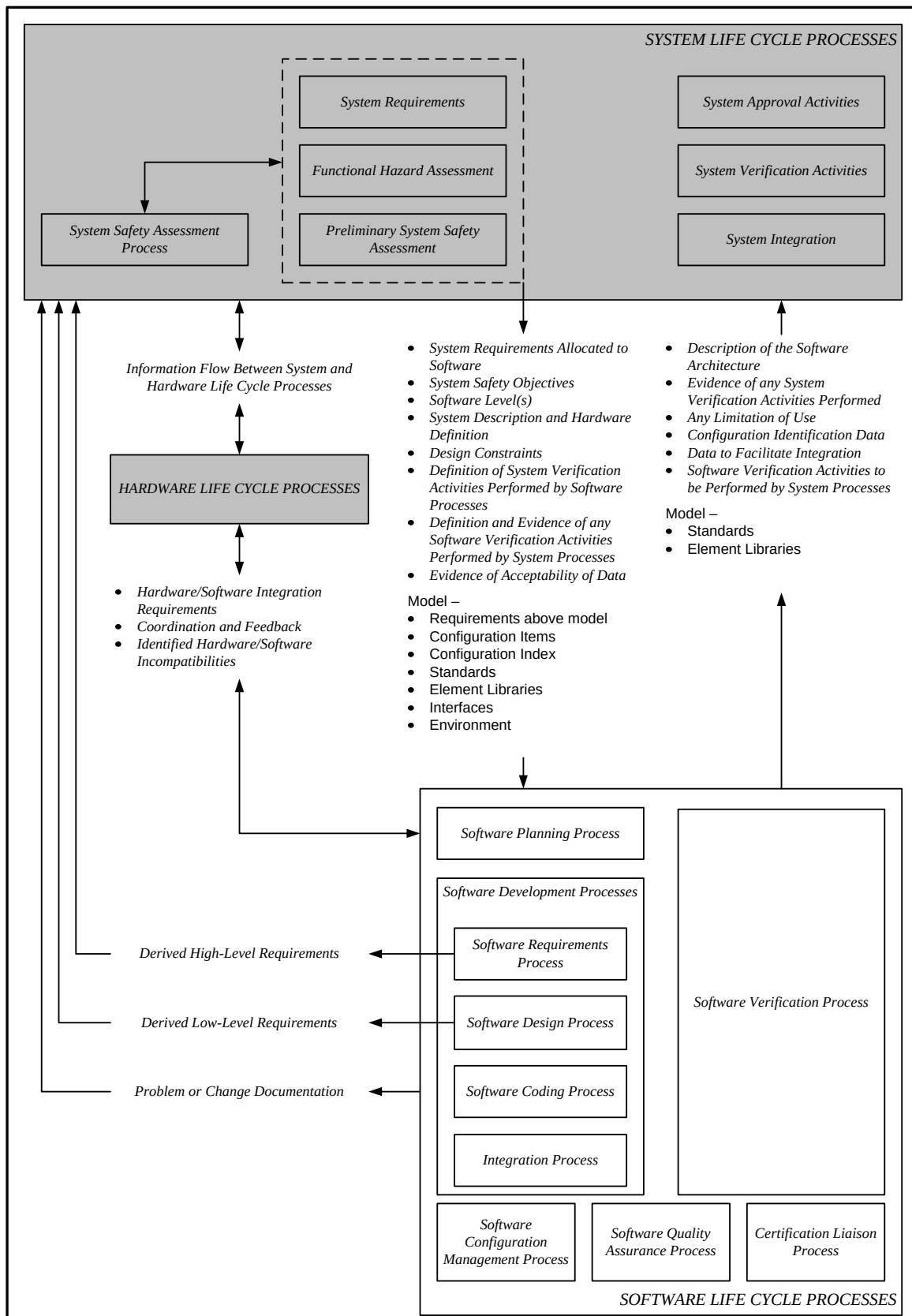


FIGURE MB.2-1: INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES

MB.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. *Software 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 to the system 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.*
- h. *Evidence of software verification activities performed by the system life cycle processes, if any.*

If system processes provide a Specification Model or Design Model to the software processes, then the following data should be provided as inputs to the software processes:

- i. *Requirements from which the model was developed.*
- j. *Model configuration items (files or data that represent the model).*
- k. *Model standards describing the modeling techniques (see MB.11.23).*
- l. *Model element libraries.*
- m. *Model and system interfaces description.*
- n. *Configuration index of the model configuration items.*
- o. *Modeling development environment and user's manuals.*
- p. *Any data from verification and/or validation activities performed at system level that may be used to satisfy verification objectives as defined in section MB.6.*

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

MB.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.*

When the system processes provide a Specification Model or a Design Model to the software processes, modifications to the following data should be coordinated between the system development and software development processes. This data should be under the appropriate configuration control category for the software level:

- j. Model standards describing the modeling techniques (see MB.11.23).
- k. Model element libraries.

MB.2.2.3 Information Flow between Software Processes and Hardware Processes

Section 2.2.3 of ED-12C is unchanged.

MB.2.3 SYSTEM SAFETY ASSESSMENT PROCESS AND SOFTWARE LEVEL

Section 2.3 of ED-12C is unchanged.

MB.2.4 ARCHITECTURAL CONSIDERATIONS

Section 2.4 of ED-12C is unchanged.

MB.2.5 SOFTWARE CONSIDERATIONS IN SYSTEM LIFE CYCLE PROCESSES

Section 2.5 of ED-12C is unchanged.

MB.2.5.1 Parameter Data Items

Section 2.5.1 of ED-12C is unchanged.

MB.2.5.2 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 certification 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 MB.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 MB.12.1.1.

Guidance for user-modifiable software includes:

- a. *The user-modifiable software should not adversely affect safety, operational capabilities, flight crew workload, 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 certification 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 software 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, 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 aircraft is not compromised.*

MB.2.5.3 Commercial-Off-The-Shelf Software

COTS software included in airborne systems or equipment should satisfy the objectives of this document.

*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 MB.12.1.4, *Upgrading a Development Baseline*, and section MB.12.3.4, *Product Service History*, may be relevant in this instance.*

MB.2.5.4 Option-Selectable Software

Some airborne 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 MB.4.2.h, ED-12C sections 5.2.4, and ED-12C section 6.4.4.3.d.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.

MB.2.5.5 Field-Loadable Software

Section 2.5.5 of ED-12C is unchanged.

MB.2.5.6 Software Considerations in System Verification

Section 2.5.6 of ED-12C is unchanged.

MB.2.6 SYSTEM CONSIDERATIONS IN SOFTWARE LIFE CYCLE PROCESSES

Section 2.6 of ED-12C is unchanged.

MB.3.0**SOFTWARE LIFE CYCLE**

Section 3.0 of ED-12C is unchanged.

MB.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 MB.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 MB.5 describes the software development processes.*
- c. *The integral processes that ensure the correctness and 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 certification 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 MB.6 through MB. 9 describe the integral processes.*

MB.3.2 SOFTWARE LIFE CYCLE DEFINITION

Section 3.2 of ED-12C is unchanged.

MB.3.3 TRANSITION CRITERIA BETWEEN PROCESSES

Section 3.3 of ED-12C is unchanged.

MB.4.0

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 MB.A-1 of Annex MB.A is a summary of the objectives and outputs of the software planning process by software level.

The applicant should ensure that the intended use of this supplement is acceptable to the appropriate certification authority. Several supplements may be applied to an applicant's software life cycle. The applicant should plan an approach that will comply with the objectives of ED-12C and those applicable objectives from the supplements. As such, all relevant supplements and the supplements' associated objectives should be considered. If the applicant's approach identifies a conflict between the objectives, then the applicant should propose a solution to the conflict in the Plan for Software Aspects of Certification (PSAC). The applicant may use a single set of software plans or multiple software plans addressing the different supplements. Using a single set of software plans, containing sections addressing each supplement, will provide an integrated approach with a clear and comprehensive plan.

As supplements only extend the guidance from ED-12C for a specific technology, the applicant should first consider all of the ED-12C objectives, and then the supplement's objectives, and lastly any other additional considerations. One approach is to consolidate all objectives of ED-12C and the applicable supplements, and for each objective provide a statement of how compliance will be achieved, along with identifying any applicable life cycle data items. Because the objective numbering scheme is unique between the Annex A Tables of ED-12C and the Annex A Tables of the supplements, the identification of the objectives and their document sources will be clear and unambiguous.

Specifically related to model-based development, it may be difficult to separate system and software life cycle data. The Specification Model or the Design Model may originate from the system requirements and design process. Regardless of where these models originate, the guidance of this supplement applies and the objectives of this supplement should be satisfied.

MB.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 software 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 software level(s) are defined (see MB.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 MB.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 MB.4.4).*
- d. Additional considerations, such as those discussed in section MB.12, have been addressed, if necessary.*
- e. Software development standards, including the Software Model Standards, consistent with the system safety objectives for the software to be produced are defined (see MB.4.5).*
- f. Software plans that comply with sections MB.4.3 and MB.11 have been produced.*
- g. Development and revision of the software plans are coordinated (see MB.4.3).*

MB.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. See also MB.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 MB.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 MB.5.2.3) should be specified in the software plans and standards.*
- j. *When parameter data items are planned, the following should be addressed:*
 1. *The way that parameter data items are used.*
 2. *The software level of the parameter data items.*
 3. *The processes to develop, verify, and modify parameter 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.*
- m. *The software planning process should identify the software life cycle data represented by each model, for example, requirements data or design data. Each development model should be categorized as a Specification Model or a Design Model. The software planning process should also identify the methods (that is, review, analysis, and simulation as in section MB.6.8.1) to be used to verify the model. This includes the definition of the model coverage analysis criteria as introduced in section MB.6.7.1 for Design Models. When multiple models are used, the software planning process should address each model.*
- n. *A model is characterized by the modeling technique that is used and by the type of software life cycle data they will represent. Each modeling technique used, and rationale for its suitability to the type of information expressed by the Specification Model or Design Model, should be described in the Software Model Standards (see MB.11.23). Software Model Standards should be developed for each type of the model being developed (that is, a Specification Model or a Design Model).*
- o. *For Design Models, simulation may be used in combination with testing and appropriate analysis to achieve objectives related to the verification of the Executable Object Code. In that case, the planning process should address the specific constraints defined in section MB.6.8.2.*

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

MB.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 Certification (see MB.11.1) serves as the primary means for communicating the proposed development methods to the certification authority for agreement, and defines the means of compliance with this document.*
- *The Software Development Plan (see MB.11.2) defines the software life cycle(s), software development environment, and the means by which the software development process objectives will be satisfied.*
- *The Software Verification Plan (see MB.11.3) defines the means by which the software verification process objectives will be satisfied.*
- *The Software Configuration Management Plan (see MB.11.4) defines the means by which the software configuration management process objectives will be satisfied.*
- *The Software Quality Assurance Plan (see ED-12C section 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 certified product. Such changes may be as a result of feedback from other processes and may cause a change to the software plans.*

MB.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 MB.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 MB.12.2).*

MB.4.4.1 Software Development Environment

Section 4.4.1 of ED-12C is unchanged.

MB.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 software 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 ED-12C 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 ED-12C section 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 MB.6 and MB.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.*

MB.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 MB.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.*

MB.4.4.4 Simulation Environment

Model simulation environment planning defines the methods, tools, procedures, and operating environment that may be used when performing selected verification activities of the outputs of the model development process using a model simulator. Activities include:

- a. Determination of whether the model simulator needs to be qualified should be evaluated according to the criteria given in section MB.12.2.
- b. Capabilities and limitations of the model simulator with regards to its intended use and their effects on the ability to detect errors and verify functionality should be addressed. Detection of errors not capable of being found by model simulation should be provided by other software verification process activities and the corresponding method(s) specified in the Software Verification Plan.
- c. Verification planning should assess the effects of simulation environment changes and consider re-verification consistent with the guidance of sections MB.6 and MB.12.1.3.

MB.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, the Software Code Standards, and the Software Model 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 MB.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.*

MB.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 are chosen that enable the objectives of this document to be satisfied.*
- b. *Software life cycle processes can be applied consistently.*
- c. *Each process produces evidence that its outputs can be traced to their activity and inputs, showing the degree of independence of the activity, the environment, and the methods to be used.*
- d. *The outputs of the software planning process are consistent and comply with section MB.11.*

MB.5.0

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 MB.4) and the Software Development Plan (see MB.11.2). Table MB.A-2 of Annex MB.A is a summary of the objectives and outputs of the software development processes by software level. The software development processes are:

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

Software development processes produce one or more levels of software requirements, which may be expressed within models. 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. Based on the definition of Specification Model and Design Model, high-level requirements and at least one level of low-level requirements always exist. Therefore, this supplement does not allow 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.

Requirements from which the model is developed should be external to the model. Those requirements should provide details and constraints to enable model development and verification activities.

All objectives and activities related to high-level requirements contained in this section are applicable to the requirements contained in Specification Models and to the requirements from which Design Models are developed.

All objectives and activities related to software architecture and low-level requirements contained in this section are applicable to the software architecture and requirements contained in Design Models.

NOTE: *As stated in ED-12C section 2.6, certification credit may be sought from system life cycle processes for the satisfaction, or partial satisfaction, of the software objectives as defined in this supplement.*

A model cannot be classified as both a Specification Model and a Design Model.

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. Thus, Specification Models and Design Models may contain 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.

Model elements that do not contribute to the representation of any software requirement and are not inputs to a subsequent software development activity may be included in a model (for example, comment elements). These elements will not be implemented in the Executable Object Code and therefore need to be identified.

MB.5.1 SOFTWARE REQUIREMENTS PROCESS

Section 5.1 of ED-12C is unchanged.

MB.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.*
- c. *In the case where high-level requirements are expressed by a Specification Model, all model elements that do not represent software requirements and are not inputs to a subsequent software development process or activity are identified, for example, a comment block.*

NOTE: *Although there is no specific objective to verify that such elements not representing requirements were properly identified, the fulfillment of the objective defined in item c is ensured through the verification objectives of section MB.6.3.1.*

MB.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 processes, and the Software Development Plan, the Software Requirements Standards, and the Software Model Standards (if Specification Models are used) 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 MB.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, and be verifiable and consistent. When high-level requirements are expressed by a Specification Model, this model should conform to the Software Model Standards (see MB.11.23), and be verifiable and consistent.*
- f. *The high-level requirements should be stated in quantitative terms with tolerances where applicable.*
- g. *The high-level requirements should not describe design or verification detail except for specified and justified design constraints.*
- h. *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 parameter data items are planned, the high-level requirements should describe how any parameter 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 parameter data item elements should be consistent with the structure of the parameter data item and the attributes of its data elements.*
- k. When high-level requirements are expressed by a Specification Model, all model elements should be categorized as described in the Software Model Standards (see MB.11.23) by either one of the following:
 - 1. Model elements that represent high-level requirements, including derived requirements.
 - 2. Model elements that do not represent high-level requirements (for example elements used to introduce comments).
- l. When a Design Model is planned, high-level requirements for that model should be adequate to support implementation and verification of the Design Model. In particular, the following apply:
 - 1. High-level requirements should contain the level of detail that enables the development and verification of the Design Model.
 - 2. High-level requirements should provide a functional description of the software functionality.

NOTE: *Functional description provides a structured description of software functionality, including interrelationship and interfaces with the software environment, without prescribing the software architecture. Functional description is not software architecture; functionality is allocated to software components as part of software architecture design process.*

MB.5.2 SOFTWARE DESIGN PROCESS

Section 5.2 of ED-12C is unchanged.

MB.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.*
- c. In the case where low-level requirements and/or architecture are expressed by a Design Model, all model elements that do not represent software requirements or software architecture and are not inputs to a subsequent software development process or activity are identified, for example, a comment block.

NOTE: *Although there is no specific objective to verify that such elements not representing requirements and/or software architecture were properly identified, the fulfillment of the objective defined in item c is ensured through the verification objectives of section MB.6.3.2 and/or MB.6.3.3.*

MB.5.2.2 Software Design Process Activities

The software design process inputs are the Software Requirements Data, the Software Development Plan, the Software Design Standards, and the Software Model Standards (if Design Models are used). 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 MB.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. When low-level requirements or software architecture are expressed by a Design Model, this model should conform to the Software Model Standards (see MB.11.23) 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 software level assignment for some components of the software. In such cases, additional data should be defined as derived requirements and provided to the system 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.*
- h. *When low-level requirements and/or architecture are expressed by a Design Model, all model elements should be categorized as described in the Software Model Standards (see MB.11.23) by either one of the following:*
 1. *Model elements that represent low-level requirements, including derived requirements, or architecture.*
 2. *Model elements that do not represent low-level requirements or architecture (for example, elements used to introduce comments).*

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.*

MB.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 software 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 MB.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.*

MB.5.2.4 Designing for Deactivated Code

Section 5.2.4 of ED-12C is unchanged.

MB.5.3 SOFTWARE CODING PROCESS

Section 5.3 of ED-12C is unchanged.

MB.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 MB.11.16) in the integration process to develop the integrated system or equipment.

MB.5.4.1 Integration Process Objectives

Section 5.4.1 of ED-12C is unchanged.

MB.5.4.2 Integration Process Activities

Section 5.4.2 of ED-12C is unchanged.

MB.5.5 SOFTWARE DEVELOPMENT PROCESS TRACEABILITY

The activities related to traceability in ED-12C section 5.5 are fully applicable.

When using model-based development, identification of requirements as per the method defined in the Software Model Standards (see MB.11.23.e and MB.11.23.f) should be used for bi-directional traceability. Means for this traceability should also be defined in the Software Model Standards (see MB.11.23.e).

The traceability for model-based development is no different than for development using traditional methods per ED-12C. Since functional requirements are implemented using combinations of model elements, these combinations should therefore be used for bi-directional traceability.

MB.6.0

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 MB.4) and the Software Verification Plan (see MB.11.3). See section MB.4.6 for the verification of the outputs of 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.

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

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

- *Verification of 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.*

All objectives and activities related to high-level requirements contained in this section are applicable to the requirements contained in Specification Models and requirements from which Design Models are developed.

All objectives and activities related to software architecture and low-level requirements contained in this section are applicable to the software architecture and requirements contained in Design Models.

NOTE: *As stated in ED-12C section 2.6, certification credit may be sought from system life cycle processes for the satisfaction, or partial satisfaction, of the software objectives as defined in this supplement.*

MB.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 of requirements. If the output of the design process is a sequence of Design Models which express lower levels of requirements, each Design Model satisfies the requirements from which it is developed.*
- 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 properly to abnormal inputs and conditions.*
- f. *The means used to perform this verification are technically correct and complete for the software level.*

MB.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 MB.11.13), the Software Verification Results (see MB.11.14), and the associated Trace Data (see MB.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 airborne 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 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.*

MB.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 on 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.

If simulation is being used to verify the completeness and correctness of the requirements contained in the model, then section MB.6.8.1 applies in addition to all of section MB.6.3.

MB.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 reviews 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. When high-level requirements are expressed by a model, the objective is to ensure that the Software Model Standards were followed during the software requirement 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.*

MB.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 reviews 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. This objective is also to provide confidence in detecting the presence of unintended functionality in the Design Model.*
- 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. When low-level requirements are expressed by a model, the objective is to ensure that the Software Model 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.*

NOTE: *When high-level requirements are expressed by a Specification Model, this objective also aims at ensuring that no low-level requirement traces to model elements that do not represent high-level requirements (see MB.5.1.1.c).*

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

MB.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 reviews 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 software level, it should also be confirmed that the higher software level component has appropriate protection mechanisms in place to protect itself from potential erroneous inputs from the lower software 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 restrictions and design constraint rules. When software architecture is expressed by a model, the objective is to ensure that the Software Model Standards were followed during the software design process and that deviations from the standards are justified.
- f. Partitioning integrity: The objective is to ensure that partitioning breaches are prevented.

MB.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 and the software architecture, and conformance to the Software Code Standards. These reviews 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.
- b. Compliance with the software architecture: The objective is to ensure that the Source Code matches the data flow and control flow defined in the software architecture.
- c. Verifiability: The objective is to ensure the Source Code does not contain statements and structures that cannot be verified and that the code does not have to be altered to test it.
- d. Conformance to standards: The objective is to ensure that the Software Code Standards were followed during the development of the code, for example, complexity restrictions and code constraints. Complexity includes the degree of coupling between software components, the nesting levels for control structures, and the complexity of logical or numeric expressions. This analysis also ensures that deviations to the standards are justified.
- e. Traceability: The objective is to ensure that the low-level requirements were developed into Source Code.
NOTE: When low-level requirements are expressed by a Design Model, this objective also aims at ensuring that no Source Code traces to model elements that do not represent low-level requirements (see MB.5.2.1.c).
- f. Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, memory usage, fixed point arithmetic overflow and resolution, floating-point arithmetic, resource contention and limitations, worst-case execution timing, exception handling, use of uninitialized variables, cache management, unused variables, and data corruption due to task or interrupt conflicts. The compiler (including its options), the linker (including its options), and some hardware features may have an impact on the worst-case execution timing and this impact should be assessed.

MB.6.3.5 Reviews and Analyses of the Outputs of the Integration Process

Section 6.3.5 of ED-12C is unchanged.

MB.6.4 SOFTWARE TESTING

Section 6.4 of ED-12C is unchanged.

If simulation is being used as part of Executable Object Code verification, then section MB.6.8.2 applies in addition to section 6.4 of ED-12C.

MB.6.5 SOFTWARE VERIFICATION PROCESS TRACEABILITY

Section 6.5 of ED-12C is unchanged.

MB.6.6 VERIFICATION OF PARAMETER DATA ITEMS

Section 6.6 of ED-12C is unchanged.

MB.6.7 MODEL COVERAGE ANALYSIS FOR DESIGN MODELS

This section only applies when Design Models are used in the software development process.

Model coverage analysis determines which requirements expressed by the model were not exercised by verification based on the requirements from which the model was developed. Model coverage analysis is a means to assess thoroughness of the model verification activities.

This analysis supports the detection of unintended functionality (with respect to the requirements from which the model was developed) in the Design Model, where coverage of the requirements from which the model was developed has been achieved by the verification cases.

Model coverage analysis is different than structural coverage analysis and therefore model coverage analysis does not eliminate the need to achieve the objectives of structural coverage analysis per ED-12C section 6.4.4.2.

Model coverage analysis should use the outputs (cases, procedures, and/or results) from one or more verification techniques: simulation, testing, and/or other appropriate techniques. Model coverage analysis does not eliminate the need for traceability analysis between requirements from which the model was developed and the model. Traceability analysis can also support model coverage analysis if the Trace Data used for this purpose are adequate (for example, the granularity of the Trace Data is appropriate) to assess the model coverage analysis criteria, as defined during the planning process.

Since model coverage analysis can be supported by different verification techniques, for simplicity purposes, the text in this section will refer to verification cases as one or more of the following:

- Simulation cases.
- Test cases.
- Cases produced through other verification techniques such as formal methods.

Model coverage analysis activities include:

- a. Model coverage analysis should be performed using requirements-based verification cases as described in ED-12C section 6.4.2 using the requirements from which the Design Model was developed.
- b. Model coverage analysis should confirm the degree of coverage of the Design Model by the verification cases based on the requirements from which the Design Model was developed in accordance with the defined coverage criteria (see MB.6.7.1). In the event complete model coverage cannot be achieved, model coverage resolution should be performed according to section MB.6.7.2.
- c. Model coverage analysis should be supplemented with additional verification cases based on derived requirements contained in the Design Model (see MB.6.7.1).

MB.6.7.1 Model Coverage Analysis Criteria

As the intent of model coverage analysis is to detect unintended function in the Design Model, it is recognized that various methods are available to perform the model coverage analysis. In particular, the model coverage analysis criteria can be defined in multiple ways but should comply with ED-12C sections 6.4.2.1 and 6.4.2.2. Therefore, the criteria should be defined during the planning process and indicated in the Software Verification Plan. The following shows an example of criteria that may be used to assess model coverage; however, alternative criteria may be proposed.

TABLE MB.6-1: MODEL COVERAGE CRITERIA EXAMPLE

Typical Completion Criteria	Satisfy by verification cases and justifications based on requirements from which the Design Model was developed	Satisfy by verification cases and justifications based on requirements contained in the Design Model
Coverage of all characteristics of the functionality in context, for example watchdog function triggered (See Note)	Recommended	
For state machines: coverage of all the transitions (See Note)	Recommended	
For logic equations: coverage of all the decisions (See Note)	Recommended	
Coverage of all equivalence classes and boundary/singular values for numeric data (See Note)	Recommended	Alternatively
Coverage of all derived requirements (not traceable to higher-level requirements)		Recommended

NOTE: *These criteria are compatible with ED-12C sections 6.4.2.1 and 6.4.2.2.*

MB.6.7.2 Model Coverage Analysis Resolution

Possible causes of verification deficiencies revealed by model coverage analysis, and the additional activities required to resolve the deficiencies, include:

- a. Shortcomings in requirements-based verification cases or procedures: The verification cases should be supplemented or verification procedures changed to provide the missing coverage. The method(s) used to perform the coverage analyses based on the requirements from which the Design Model was developed may need to be reviewed.
- b. Inadequacies or shortcomings in requirements from which the Design Model was developed: The requirements from which the Design Model was developed should be modified and additional verification cases developed and verification procedures executed.
- c. Derived requirements expressed by the Design Model: Appropriate verification cases should exist or be developed for the derived requirements and verification procedures should be executed to provide the missing coverage. Previously unidentified derived requirements should be identified, justified, and provided to the system processes, including system safety assessment process.
- d. Deactivated functionality expressed by the Design Model: Deactivated functionality expressed by the Design Model should be justified. For deactivated functionality expressed by a Design Model that is not intended to be realized in any configuration used within an aircraft or engine, a combination of analysis, simulation and testing should show that its realization is prevented, isolated, or eliminated. For deactivated functionality expressed by a Design Model that is only intended to be realized in certain approved configurations used within an aircraft or engine, the operational configuration needed for normal realization of these requirements should be established and additional verification cases and verification procedures developed to satisfy the required coverage objectives.
- e. Unintended functionality expressed by a Design Model: Unintended functionality indicates an error is present and the functionality should be removed from the Design Model.

MB.6.8 MODEL SIMULATION

The goal of this section is to provide guidance, including limitations, to allow the use of simulation without compromising the rigor appropriate for the software level. The use of model simulation in compliance with this supplement does not alleviate the guidance of ED-12C sections 6.3 and 6.4. When simulation is used, all objectives of high-level requirements verification, software design verification, Executable Object Code verification, and software testing verification, should still be satisfied (see MB.6.3.1, MB.6.3.2, MB.6.3.3, and MB.6.4). Model simulation may support the achievement of some of these objectives. In this case, it is necessary to ensure that model simulation is planned, developed, and performed completely and correctly to support satisfaction of those specific objectives.

For Specification Models or Design Models, simulation may be used in combination with reviews and analysis of requirements and architecture to satisfy some objectives of sections MB.6.3.1, MB.6.3.2, and MB.6.3.3. In this case, guidance on the verification of a model by means of simulation in section MB.6.8.1 should be followed.

For Design Models, simulation may be used in combination with testing and appropriate coverage analysis to satisfy objectives related to the verification of the Executable Object Code. As simulation may involve different object code and a different environment than the target application, simulation alone cannot be used to satisfy objectives related to verification of the Executable Object Code. Guidance on the verification of Executable Object Code by means of simulation in section MB.6.8.2 should be followed. However, if simulation cases are run in the target computer environment using the Executable Object Code, then they are also considered test cases and ED-12C section 6.4 applies.

MB.6.8.1 Model Simulation for Verification of the Model

The main purpose of simulation is to provide repeatable evidence of compliance of the model to the requirements from which the model was developed, satisfying the following objectives:

- Compliance to system requirements for Specification Models (see MB.6.3.1.a).
- Compliance to software high-level requirements for Design Models containing low-level requirements (see MB.6.3.2.a).
- Compliance to software high-level requirements for Design Models containing software architecture (see MB.6.3.3.a).

Several other objectives may be satisfied by simulation for Specification Models and Design Models such as:

- Accuracy and consistency (see MB.6.3.1.b and MB.6.3.2.b).
- Verifiability (see MB.6.3.1.d and MB.6.3.2.d).
- Algorithm aspects (see MB.6.3.1.g and MB.6.3.2.g).

In addition, simulation may provide evidence for Design Models containing software architecture of the following:

- Consistency (see MB.6.3.3.b).
- Verifiability (see MB.6.3.3.d).

However, simulation cannot be used to satisfy the following objectives:

- Compatibility with the target computer (see MB.6.3.1.c, MB.6.3.2.c, and MB.6.3.3.c).
- Conformance to standards (see MB.6.3.1.e, MB.6.3.2.e, and MB.6.3.3.e).
- Traceability (see MB.6.3.1.f and MB.6.3.2.f).
- Partitioning integrity (see MB.6.3.3.f).

NOTE: *Model simulation activity is not limited to the execution of the simulation cases and procedures but also encompasses the review/analysis of the requirements in order to determine which objectives can be supported by simulation as well as the review/analysis of the simulation results to confirm the objectives have been satisfied (see MB.6.8.3.2.c).*