RTCA, Inc.
1140 Connecticut Avenue, N. W., Suite 1020
Washington, D. C. 20036

SOFTWARE CONSIDERATIONS IN AIRBORNE SYSTEMS AND EQUIPMENT CERTIFICATION

RTCA/DO-178B
December 1, 1992

Prepared by:
RTCA SC-167 / EUROCAE WG-12

"Requirements and Technical Concepts for Aviation"

Copies of this document may be obtained from

RTCA, Inc.
1140 Connecticut Avenue, Northwest, Suite 1020
Washington, D. C. 20036-4001 U.S.A.

Telephone: 202-833-9339
Facsimile: 202-833-9439

Please contact RTCA for price and ordering information.

FOREWORD

This document was prepared by Special Committee 167 of RTCA, Inc. It was approved by RTCA, Inc. on December 1, 1992.

RTCA is an association of aeronautical organizations of the United States of America from both government and industry. Dedicated to the advancement of aeronautics, RTCA seeks sound technical solutions to problems involving the application of electronics and telecommunications to aeronautical operations. Its objective is the resolution of such problems by mutual agreement of its member and participating organizations.

The findings of RTCA are in the nature of recommendations to all organizations concerned. As RTCA is not an official agency of the United States Government, its recommendations may not be regarded as statements of official government policy unless so enunciated by the federal government organizations or agency having statutory jurisdiction over any matters to which the recommendations relate.

The development of these guidelines was jointly accomplished by RTCA SC-167 and the European Organisation for Civil Aviation Equipment (EUROCAE) WG-12 through a consensus process.

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

TABLE OF CONTENTS

Page

LIST OF FIGURES AND TABLES

FIGURES

Page

TABLES

Page

THIS PAGE INTENTIONALLY LEFT BLANK

1.0 INTRODUCTION

The rapid increase in the use of software in airborne systems and equipment used on aircraft and engines in the early 1980s resulted in a need for industry-accepted guidance for satisfying airworthiness requirements. DO-178, "Software Considerations in Airborne Systems and Equipment Certification," was written to satisfy this need.

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

1.1 Purpose

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

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

1.2 Scope

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

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

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

1.3 Relationship to Other Documents

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

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

1.4 How to Use This Document

These points need to be noted when using this document:

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

1.5 Document Overview

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

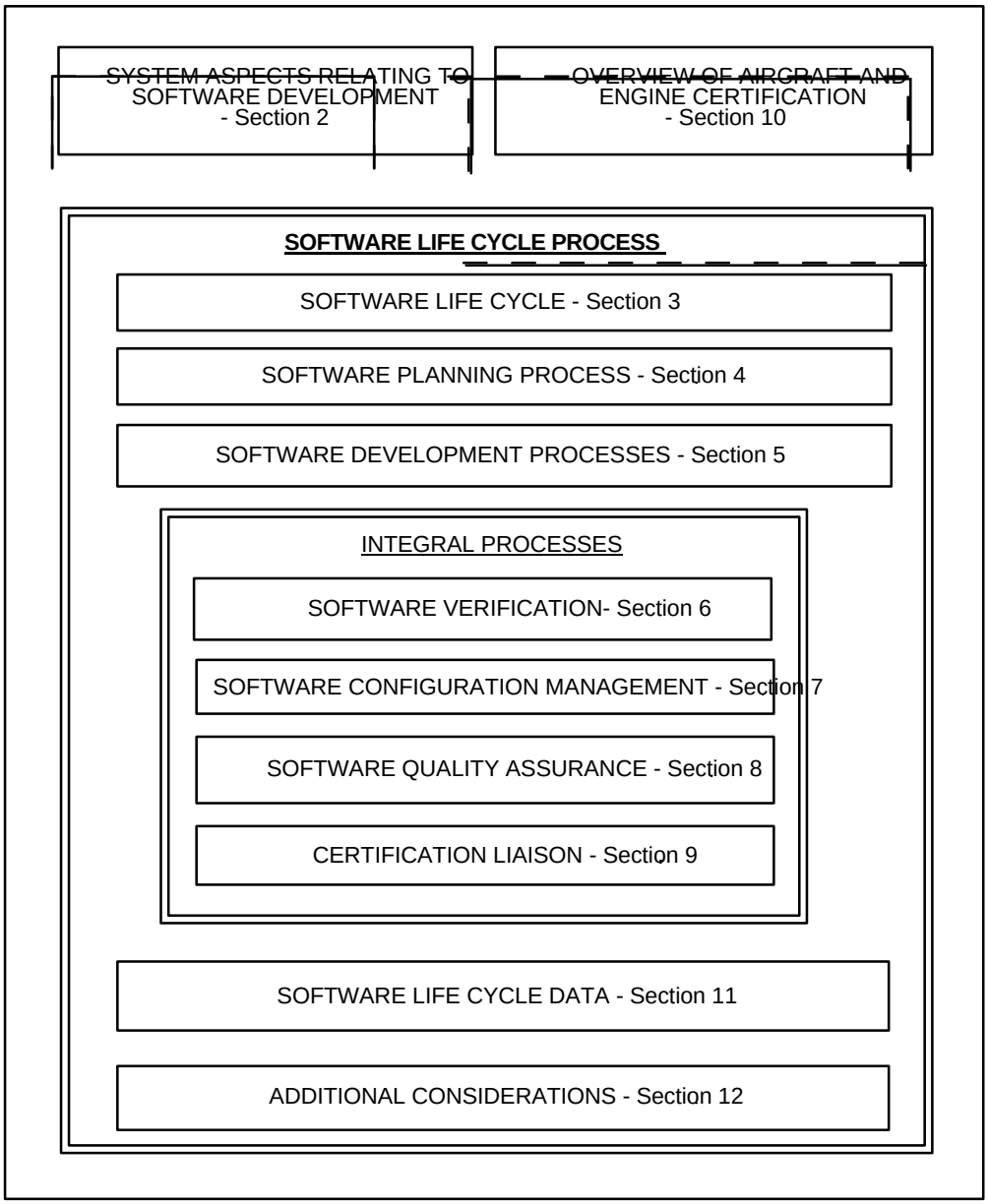

FIGURE 1-1
DOCUMENT OVERVIEW

THIS PAGE INTENTIONALLY LEFT BLANK

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. Discussed are:

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

2.1 Information Flow Between System and Software Life Cycle Processes

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

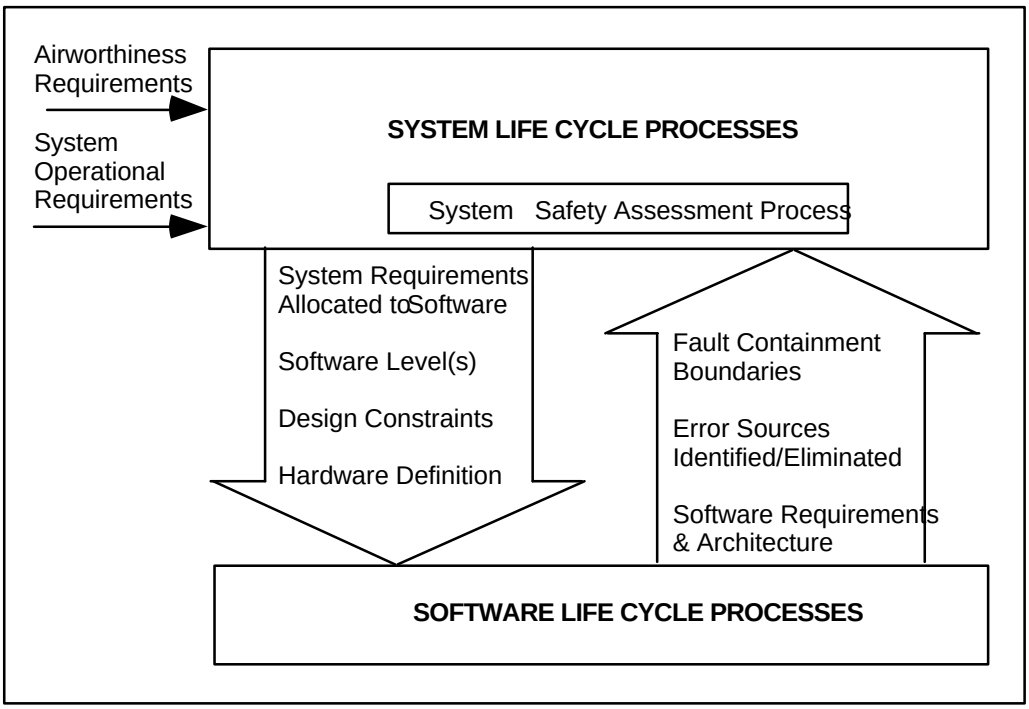

FIGURE 2-1
SYSTEM SAFETY-RELATED INFORMATION FLOW BETWEEN SYSTEM AND SOFTWARE LIFE CYCLE PROCESSES

Note: *At the time of publication of this document, guidelines for the system life cycle processes were under development by an international committee. While every attempt was made to keep the inter-process information flows and definitions compatible, some differences may*

exist between the final published documents. Any differences will be reconciled in future revisions of the documents.

2.1.1 Information Flow from System Processes to Software Processes

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

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

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

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

2.1.2 Information Flow from Software Processes to System Processes

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

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

2.2 Failure Condition and Software Level

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

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

2.2.1 Failure Condition Categorization

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

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

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

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

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

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

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

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

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

2.2.2 Software Level Definitions

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

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

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

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

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

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

2.2.3 Software Level Determination

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

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

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

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

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

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

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

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

2.3 System Architectural Considerations

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

2.3.1 Partitioning

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

Guidance for partitioning includes:

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

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

   (2) Control coupling: vulnerability to external access.

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

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

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

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

2.3.2 Multiple-Version Dissimilar Software

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

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

The degree of dissimilarity and hence the degree of protection is not usually measurable. Probability of loss of system function will increase to the extent that the safety monitoring associated with dissimilar software versions detects actual errors or experiences transients that exceed comparator threshold limits. Dissimilar software versions are usually used, therefore, as a means of providing additional protection after the software verification process objectives for the software level, as described in section 6, have been satisfied. Dissimilar software verification methods may be reduced from those used to verify single version software if it can be shown that

the resulting potential loss of system function is acceptable as determined by the system safety assessment process.

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

2.3.3 Safety Monitoring

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.5 System Design Considerations for Field-Loadable Software

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

System safety considerations relating to field-loadable software include:

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

Guidance for field-loadable software includes:

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

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

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

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

2.6 System Requirements Considerations for Software Verification

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

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

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

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

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

2.7 Software Considerations in System Verification

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

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

THIS PAGE INTENTIONALLY LEFT BLANK

3.0 SOFTWARE LIFE CYCLE

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

3.1 Software Life Cycle Processes

The software life cycle processes are:

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

3.2 Software Life Cycle Definition

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

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

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

The processes of a software life cycle may be iterative, that is, entered and re-entered. The timing and degree of iteration varies due to the incremental development of system functions, complexity,

requirements development, hardware availability, feedback to previous processes, and other attributes of the project.

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

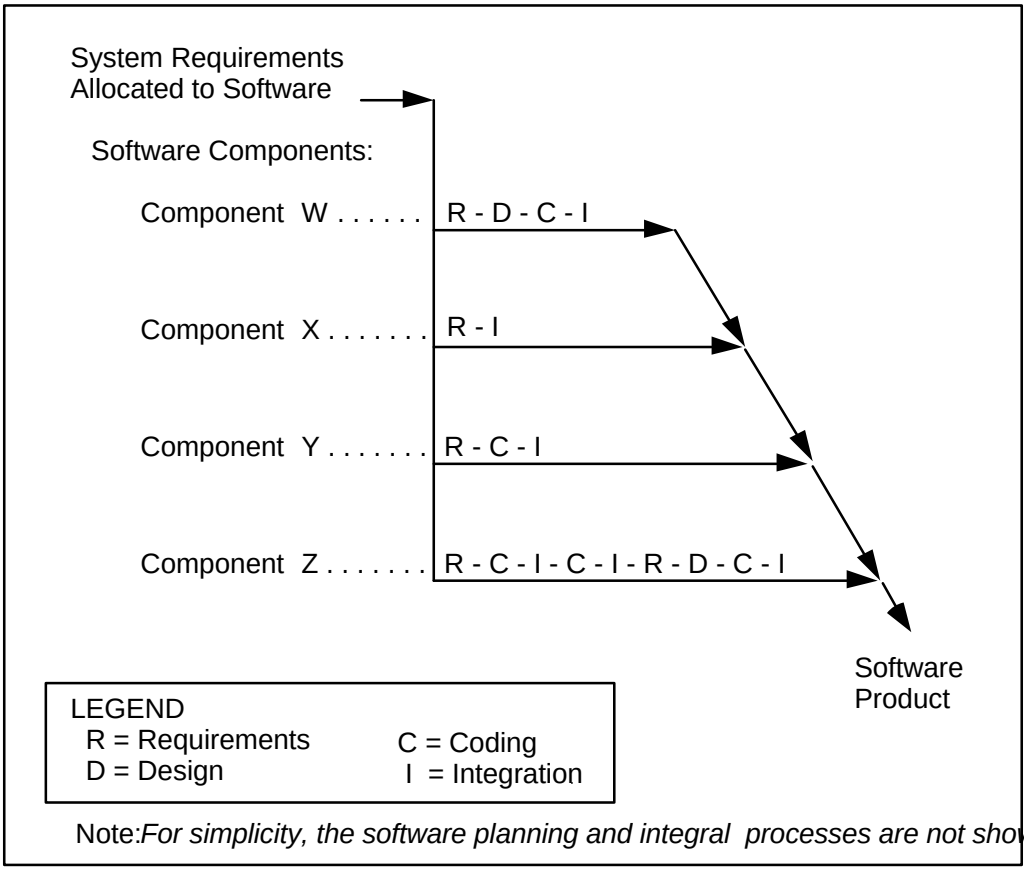

FIGURE 3-1
EXAMPLE OF A SOFTWARE PROJECT
USING FOUR DIFFERENT DEVELOPMENT SEQUENCES

3.3 Transition Criteria Between Processes

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

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

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

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

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

4.1 Software Planning Process Objectives

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

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

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

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

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

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

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

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

4.2 Software Planning Process Activities

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

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

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

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

d. The software planning process should provide coordination between the software development and integral processes to provide consistency among strategies in the software plans.e. The software planning process should include a means to revise the software plans as a project progresses.

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

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

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

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

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

4.3 Software Plans

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

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

Guidance for the software plans includes:

a. The software plans should comply with this document.

b. The software plans should define the criteria for transition between software life cycle processes by specifying:(1) The inputs to the process, including feedback from other processes.

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

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

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

4.4 Software Life Cycle Environment Planning

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

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

The considerations presented above may affect:

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

4.4.1 Software Development Environment

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

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

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

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

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

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

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

4.4.2 Language and Compiler Considerations

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

a. Some compilers have features intended to optimize performance of the object code. If the test cases give coverage consistent with the software level, the correctness of the

optimization need not be verified. Otherwise, the impact of these features on structural coverage analysis should be determined following the guidelines of subparagraph 6.4.4.2.

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

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

4.4.3 Software Test Environment

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

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

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

4.5 Software Development Standards

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

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

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

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

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

4.6 Review and Assurance of the Software Planning Process

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

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

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

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

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

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

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

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

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

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

5.1 Software Requirements Process

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

5.1.1 Software Requirements Process Objectives

The objectives of the software requirements process are:

a. High-level requirements are developed.

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

5.1.2 Software Requirements Process Activities

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

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

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

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

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

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

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

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

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

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

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

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

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

5.2 Software Design Process

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

5.2.1 Software Design Process Objectives

The objectives of the software design process are :

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

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

5.2.2 Software Design Process Activities

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

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

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

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

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

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

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

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

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

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

5.2.3 Designing for User-Modifiable Software

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

Guidance for designing user-modifiable software includes:

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

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

5.3 Software Coding Process

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

5.3.1 Software Coding Process Objectives

The objective of the software coding process is:

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

5.3.2 Software Coding Process Activities

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

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

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

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

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

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

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

5.4 Integration Process

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

5.4.1 Integration Process Objectives

The objective of the integration process is:

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

5.4.2 Integration Process Activities

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

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

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

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

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

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

5.4.3 Integration Considerations

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

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

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

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

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

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

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

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

5.5 Traceability

Traceability guidance includes:

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

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

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

THIS PAGE INTENTIONALLY LEFT BLANK

6.0 SOFTWARE VERIFICATION PROCESS

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

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

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

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

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

6.1 Software Verification Process Objectives

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

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

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

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

d. The Executable Object Code satisfies the software requirements.

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

6.2 Software Verification Process Activities

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

The inputs to the software verification process include the system requirements, the software requirements and architecture, traceability data, Source Code, Executable Object Code, and the Software Verification Plan.

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

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

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

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

Guidance for the software verification activities includes:

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

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

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

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

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

6.3 Software Reviews and Analyses

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

6.3.1 Reviews and Analyses of the High-Level Requirements

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

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

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

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

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

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

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

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

6.3.2 Reviews and Analyses of the Low-Level Requirements

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

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

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

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

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

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

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

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

6.3.3 Reviews and Analyses of the Software Architecture

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

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

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

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

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

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

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

6.3.4 Reviews and Analyses of the Source Code

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

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

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

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

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

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

f. Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, fixed point arithmetic overflow and resolution, resource contention, worst-case execution timing, exception handling, use of uninitialized

variables or constants, unused variables or constants, and data corruption due to task or interrupt conflicts.

6.3.5 Reviews and Analyses of the Outputs of the Integration Process

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

a. Incorrect hardware addresses.

b. Memory overlaps.

c. Missing software components.

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

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

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

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

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

6.4 Software Testing Process

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

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

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

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

To satisfy the software testing objectives:

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

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

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

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

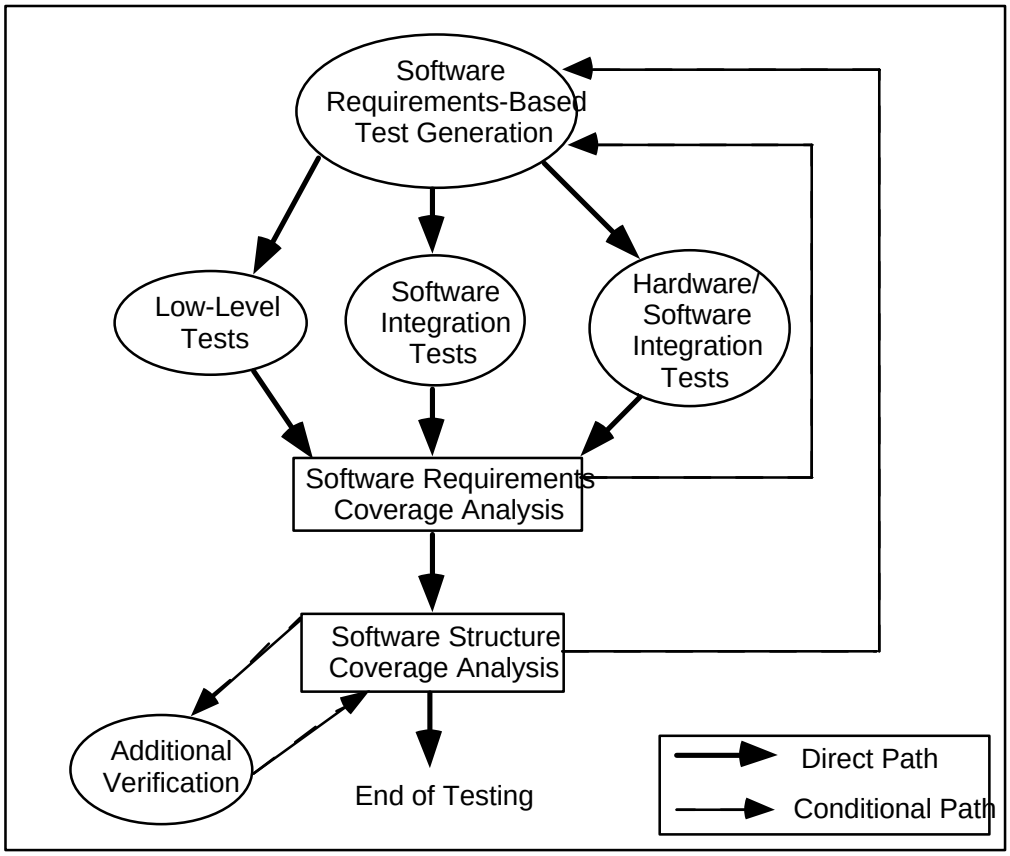

FIGURE 6-1
SOFTWARE TESTING PROCESS

6.4.1 Test Environment

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

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

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

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

6.4.2 Requirements-Based Test Case Selection

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

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

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

6.4.2.1 Normal Range Test Cases

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

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

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

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

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

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

6.4.2.2 Robustness Test Cases

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

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

b. System initialization should be exercised during abnormal conditions.

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

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

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

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

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

6.4.3 Requirements-Based Testing Methods

Requirements-based testing methods consist of methods for requirements-based hardware/software integration testing, requirements-based software integration testing, and requirements-based low-

level testing. With the exception of hardware/software integration testing, these methods do not prescribe a specific test environment or strategy. Guidance includes:

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

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

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

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

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

Typical errors revealed by this testing method include:

- Failure of an algorithm to satisfy a software requirement.
- Incorrect loop operations.
- Incorrect logic decisions.

- Failure to process correctly legitimate combinations of input conditions.
- Incorrect responses to missing or corrupted input data.
- Incorrect handling of exceptions, such as arithmetic faults or violations of array limits.
- Incorrect computation sequence.
- Inadequate algorithm precision, accuracy or performance.

6.4.4 Test Coverage Analysis

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

6.4.4.1 Requirements-Based Test Coverage Analysis

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

a. Test cases exist for each software requirement.

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

6.4.4.2 Structural Coverage Analysis

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

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

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

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

6.4.4.3 Structural Coverage Analysis Resolution

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

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

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

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

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

THIS PAGE INTENTIONALLY LEFT BLANK

7.0 SOFTWARE CONFIGURATION MANAGEMENT PROCESS

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

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

7.1 Software Configuration Management Process Objectives

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

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

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

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

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

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

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

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

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

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

7.2 Software Configuration Management Process Activities

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

7.2.1 Configuration Identification

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

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

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

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

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

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

7.2.2 Baselines and Traceability

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

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

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

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

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

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

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

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

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

7.2.3 Problem Reporting, Tracking and Corrective Action

The objective of problem reporting, tracking and corrective action is to record process non-compliance with software plans and standards, to record deficiencies of outputs of software life cycle processes, to record anomalous behavior of software products, and to ensure resolution of these problems.

Note: *Software life cycle process and software product problems may be recorded in separate problem reporting systems.*

Guidance includes:

a. A problem report should be prepared that describes the process non-compliance with plans, output deficiency, or software anomalous behavior, and the corrective action taken, as defined in subsection 11.17.

b. Problem reporting should provide for configuration identification of affected configuration item(s) or definition of affected process activities, status reporting of problem reports, and approval and closure of problem reports.

c. Problem reports that require corrective action of the software product or outputs of software life cycle processes should invoke the change control activity.

Note: *The problem reporting and change control activities are related.*

7.2.4 Change Control

The objective of the change control activity is to provide for recording, evaluation, resolution and approval of changes throughout the software life cycle. Guidance includes:

a. Change control should preserve the integrity of the configuration items and baselines by providing protection against their change.

b. Change control should ensure that any change to a configuration item requires a change to its configuration identification.

c. Changes to baselines and to configuration items under change control should be recorded, approved, and tracked. Problem reporting is related to change control, since resolution of a reported problem may result in changes to configuration items or baselines.

   Note: *It is generally recognized that early implementation of change control assists the control and management of software life cycle process activities.*

d. Software changes should be traced to their origin and the software life cycle processes repeated from the point at which the change affects their outputs. For example, an error discovered at hardware/software integration, that is shown to result from an incorrect design, should result in design correction, code correction and repetition of the associated integral process activities.

e. Throughout the change activity, software life cycle data affected by the change should be updated and records should be maintained for the change control activity.

The change control activity is aided by the change review activity.

7.2.5 Change Review

The objective of the change review activity is to ensure problems and changes are assessed, approved or disapproved, approved changes are implemented, and feedback is provided to affected processes through problem reporting and change control methods defined during the software planning process. The change review activity should include:

a. Confirmation that affected configuration items are configuration identified.

b. Assessment of the impact on safety-related requirements with feedback to the system safety assessment process.

c. Assessment of the problem or change, with decisions for action to be taken.

d. Feedback of problem report or change impact and decisions to affected processes.

7.2.6 Configuration Status Accounting

The objective of the status accounting activity is to provide data for the configuration management of software life cycle processes with respect to configuration identification, baselines, problem reports, and change control. The status accounting activity should include:

a. Reporting on configuration item identification, baseline identification, problem report status, change history, and release status.

b. Definition of the data to be maintained and the means of recording and reporting status of this data.

7.2.7 Archive, Retrieval and Release

The objective of the archive and retrieval activity is to ensure that the software life cycle data associated with the software product can be retrieved in case of a need to duplicate, regenerate, retest or modify the software product. The objective of the release activity is to ensure that only authorized software is used, especially for software manufacturing, in addition to being archived and retrievable. Guidance includes:

a. Software life cycle data associated with the software product should be retrievable from the approved source (for example, the developing organization or company).

b. Procedures should be established to ensure the integrity of the stored data (regardless of medium of storage) by:

   (1) Ensuring that no unauthorized changes can be made.

   (2) Selecting storage media that minimize regeneration errors or deterioration.

   (3) Exercising and/or refreshing archived data at a frequency compatible with the storage life of the medium.

   (4) Storing duplicate copies in physically separate archives that minimize the risk of loss in the event of a disaster.

c. The duplication process should be verified to produce accurate copies, and procedures should exist that ensure error-free copying of the Executable Object Code.

d. Configuration items should be identified and released prior to use for software manufacture and the authority for their release should be established. As a minimum, the components of the software product loaded into the airborne system or equipment (which includes the Executable Object Code and may also include associated media for software loading) should be released.

   Note: *Release is generally also required for the data that defines the approved software for loading into the airborne system or equipment. Definition of that data is outside the scope of this document, but may include the Software Configuration Index.*

e. Data retention procedures should be established to satisfy airworthiness requirements and enable software modifications.

   Note: *Additional data retention considerations may include items such as business needs and future certification authority reviews, which are outside the scope of this document.*

7.2.8 Software Load Control

The objective of the software load control activity is to ensure that the Executable Object Code is loaded into the airborne system or equipment with appropriate safeguards. Software load control refers to the process by which programmed instructions and data are transferred from a master memory device into the airborne system or equipment. For example, methods may include (subject

to approval by the certification authority) the installation of factory pre-programmed memory devices or *in situ* re-programming of the airborne system or equipment using a field loading device. Whichever method is used, software load control should include:

a. Procedures for part numbering and media identification that identify software configurations that are intended to be approved for loading into the airborne system or equipment.

b. Whether the software is delivered as an end item or is delivered installed in the airborne system or equipment, records should be kept that confirm software compatibility with the airborne system or equipment hardware.

Note: *Additional guidance on software load control is provided in subsection 2.5.*

7.2.9 Software Life Cycle Environment Control

The objective of software life cycle environment control is to ensure that the tools used to produce the software are identified, controlled, and retrievable. The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (subsection 11.15). Guidance includes:

a. Configuration identification should be established for the Executable Object Code (or equivalent) of the tools used to develop, control, build, verify, and load the software.

b. The SCM process for controlling qualified tools, should comply with the objectives associated with Control Category 1 or 2 data (subsection 7.3), as specified in paragraph 12.2.3, item b.

c. Unless 7.2.9 item b applies, the SCM process for controlling the Executable Object Code (or equivalent) of tools used to build and load the software (for example, compilers, assemblers, linkage editors) should comply with the objectives associated with Control Category 2 data, as a minimum.

7.3 Data Control Categories

Software life cycle data can be assigned to one of two categories: Control Category 1 (CC1) and Control Category 2 (CC2). These categories are related to the configuration management controls placed on the data. Table 7-1 defines the set of SCM process objectives associated with each control category, where • indicates that the objectives apply for software life cycle data of that category.

The tables of Annex A specify the control category for each software life cycle data item, by software level. Guidance for data control categories includes:

a. The SCM process objectives for software life cycle data categorized as CC1 should be applied according to Table 7-1.

b. The SCM process objectives for software life cycle data categorized as CC2 should be applied according to Table 7-1 as a minimum.

TABLE 7-1
SCM PROCESS OBJECTIVES ASSOCIATED WITH CC1 AND CC2 DATA

SCM Process Objective	Reference	CC1	CC2
Configuration Identification	**7.2.1**	•	•
Baselines	**7.2.2a, b, c, d, e**	•	
Traceability	**7.2.2f, g**	•	•
Problem Reporting	**7.2.3**	•	
Change Control - integrity and identification	**7.2.4a, b**	•	•

Change Control - tracking	**7.2.4c, d, e**	•	
Change Review	**7.2.5**	•	
Configuration Status Accounting	**7.2.6**	•	
Retrieval	**7.2.7a**	•	•
Protection against Unauthorized Changes	**7.2.5b(1)**	•	•
Media Selection, Refreshing, Duplication	**7.2.7b(2), (3), (4), c**	•	
Release	**7.2.7d**	•	
Data Retention	**7.2.7e**	•	•

THIS PAGE INTENTIONALLY LEFT BLANK

8.0 SOFTWARE QUALITY ASSURANCE PROCESS

This section discusses the objectives and activities of the software quality assurance (SQA) process. The SQA process is applied as defined by the software planning process (section 4) and the Software Quality Assurance Plan (subsection 11.5). Outputs of the SQA process activities are recorded in Software Quality Assurance Records (subsection 11.19) or other software life cycle data.

The SQA process assesses the software life cycle processes and their outputs to obtain assurance that the objectives are satisfied, that deficiencies are detected, evaluated, tracked and resolved, and that the software product and software life cycle data conform to certification requirements.

8.1 Software Quality Assurance Process Objectives

The SQA process objectives provide confidence that the software life cycle processes produce software that conforms to its requirements by assuring that these processes are performed in compliance with the approved software plans and standards.

The objectives of the SQA process are to obtain assurance that:

a. Software development processes and integral processes comply with approved software plans and standards.

b. The transition criteria for the software life cycle processes are satisfied.

c. A conformity review of the software product is conducted.

The applicability of the objectives by software level is specified in Table A-9 of Annex A.

8.2 Software Quality Assurance Process Activities

To satisfy the SQA process objectives:

a. The SQA process should take an active role in the activities of the software life cycle processes, and have those performing the SQA process enabled with the authority, responsibility and independence to ensure that the SQA process objectives are satisfied.

b. The SQA process should provide assurance that software plans and standards are developed and reviewed for consistency.

c. The SQA process should provide assurance that the software life cycle processes comply with the approved software plans and standards.

d. The SQA process should include audits of the software development and integral processes during the software life cycle to obtain assurance that:

   (1) Software plans are available as specified in subsection 4.2.

   (2) Deviations from the software plans and standards are detected, recorded, evaluated, tracked and resolved.

   Note: *It is generally accepted that early detection of process deviations assists efficient achievement of software life cycle process objectives.*

   (3) Approved deviations are recorded.

   (4) The software development environment has been provided as specified in the software plans.

   (5) The problem reporting, tracking and corrective action process complies with the Software Configuration Management Plan.

(6) Inputs provided to the software life cycle processes by the on-going system safety assessment process have been addressed.

Note: *Monitoring of the activities of software life cycle processes may be performed to provide assurance that the activities are under control.*

e. The SQA process should provide assurance that the transition criteria for the software life cycle processes have been satisfied in compliance with the approved software plans.

f. The SQA process should provide assurance that software life cycle data is controlled in accordance with the control categories as defined in subsection 7.3 and the tables of Annex A.

g. Prior to the delivery of software products submitted as part of a certification application, a software conformity review should be conducted.

h. The SQA process should produce records of the SQA process activities (subsection 11.19), including audit results and evidence of completion of the software conformity review for each software product submitted as part of certification application.

8.3 Software Conformity Review

The objective of the software conformity review is to obtain assurances, for a software product submitted as part of a certification application, that the software life cycle processes are complete, software life cycle data is complete, and the Executable Object Code is controlled and can be regenerated.

This review should determine that:

a. Planned software life cycle process activities for certification credit, including the generation of software life cycle data, have been completed and records of their completion are retained.

b. Software life cycle data developed from specific system requirements, safety-related requirements, or software requirements are traceable to those requirements.

c. Software life cycle data complies with software plans and standards, and is controlled in accordance with the SCM Plan.

d. Problem reports comply with the SCM Plan, have been evaluated and have their status recorded.

e. Software requirement deviations are recorded and approved.

f. The Executable Object Code can be regenerated from the archived Source Code.

g The approved software can be loaded successfully through the use of released instructions.

h. Problem reports deferred from a previous software conformity review are re-evaluated to determine their status.

i. If certification credit is sought for the use of previously developed software, the current software product baseline is traceable to the previous baseline and the approved changes to that baseline.

Note: *For post-certification software modifications, a subset of the software conformity review activities, as justified by the significance of the change, may be performed.*

9.0 CERTIFICATION LIAISON PROCESS

The objective of the certification liaison process is to establish communication and understanding between the applicant and the certification authority throughout the software life cycle to assist the certification process.

The certification liaison process is applied as defined by the software planning process (section 4) and the Plan for Software Aspects of Certification (subsection 11.1). Table A-10 of Annex A is a summary of the objectives and outputs of this process.

9.1 Means of Compliance and Planning

The applicant proposes a means of compliance that defines how the development of the airborne system or equipment will satisfy the certification basis. The Plan for Software Aspects of Certification (subsection 11.1) defines the software aspects of the airborne system or equipment within the context of the proposed means of compliance. This plan also states the software level(s) as determined by the system safety assessment process. The applicant should:

a. Submit the Plan for Software Aspects of Certification and other requested data to the certification authority for review at a point in time when the effects of changes are minimal, that is, when they can be managed within project constraints.

b. Resolve issues identified by the certification authority concerning the planning for the software aspects of certification.

c. Obtain agreement with the certification authority on the Plan for Software Aspects of Certification.

9.2 Compliance Substantiation

The applicant provides evidence that the software life cycle processes satisfy the software plans. Certification authority reviews may take place at the applicant's facilities or the applicant's suppliers' facilities. This may involve discussions with the applicant or its suppliers. The applicant arranges these reviews of the activities of the software life cycle processes and makes software life cycle data available as needed. The applicant should:

a. Resolve issues raised by the certification authority as a result of its reviews.

b. Submit the Software Accomplishment Summary (subsection 11.20) and Software Configuration Index (subsection 11.16) to the certification authority.

c. Submit or make available other data or evidence of compliance requested by the certification authority.

9.3 Minimum Software Life Cycle Data That Is Submitted to Certification Authority

The minimum software life cycle data that is submitted to the certification authority is:

- Plan for Software Aspects of Certification.
- Software Configuration Index.
- Software Accomplishment Summary.

9.4 Software Life Cycle Data Related to Type Design

Unless otherwise agreed by the certification authority, the regulations concerning retrieval and approval of software life cycle data related to the type design applies to:

- Software Requirements Data.
- Design Description.
- Source Code.
- Executable Object Code.
- Software Configuration Index.
- Software Accomplishment Summary.

10.0 OVERVIEW OF AIRCRAFT AND ENGINE CERTIFICATION

This section is an overview of the certification process for aircraft and engines with respect to software aspects of airborne systems and equipment, and is provided for information purposes only. The certification authority considers the software as part of the airborne system or equipment installed on the aircraft or engine; that is, the certification authority does not approve the software as a unique, stand-alone product.

10.1 Certification Basis

The certification authority establishes the certification basis for the aircraft or engine in consultation with the applicant. The certification basis defines the particular regulations together with any special conditions which may supplement the published regulations.

For modified aircraft or engines, the certification authority considers the impact the modification has on the certification basis originally established for the aircraft or engine. In some cases, the certification basis for the modification may not change from the original certification basis; however, the original means of compliance may not be applicable for showing that the modification complies with the certification basis and may need to be changed.

10.2 Software Aspects of Certification

The certification authority assesses the Plan for Software Aspects of Certification for completeness and consistency with the means of compliance that was agreed upon to satisfy the certification basis. The certification authority satisfies itself that the software level(s) proposed by the applicant is consistent with the outputs of the system safety assessment process and other system life cycle data. The certification authority informs the applicant of issues with the proposed software plans that need to be satisfied prior to certification authority agreement.

10.3 Compliance Determination

Prior to certification, the certification authority determines that the aircraft or engine (including the software aspects of its systems or equipment) complies with the certification basis. For the software, this is accomplished by reviewing the Software Accomplishment Summary and evidence of compliance. The certification authority uses the Software Accomplishment Summary as an overview for the software aspects of certification.

The certification authority may review at its discretion the software life cycle processes and their outputs during the software life cycle as discussed in subsection 9.2.

THIS PAGE INTENTIONALLY LEFT BLANK

11.0 SOFTWARE LIFE CYCLE DATA

Data is produced during the software life cycle to plan, direct, explain, define, record, or provide evidence of activities. This data enables the software life cycle processes, system or equipment certification, and post-certification modification of the software product. This section discusses the characteristics, form, configuration management controls, and content of the software life cycle data.

The characteristics of the software life cycle data are:

a. Unambiguous: Information is unambiguous if it is written in terms which only allow a single interpretation, aided if necessary by a definition.

b. Complete: Information is complete when it includes necessary, relevant requirements and/or descriptive material, responses are defined for the range of valid input data, figures used are labeled, and terms and units of measure are defined.

c. Verifiable: Information is verifiable if it can be checked for correctness by a person or tool.

d. Consistent: Information is consistent if there are no conflicts within it.

e. Modifiable: Information is modifiable if it is structured and has a style such that changes can be made completely, consistently, and correctly while retaining the structure.

f. Traceable: Information is traceable if the origin of its components can be determined.

In addition, this guidance applies:

g. Form: The form should provide for the efficient retrieval and review of software life cycle data throughout the service life of the airborne system or equipment. The data and the specific form of the data should be specified in the Plan for Software Aspects of Certification.

Note: *(1) The software life cycle data may take a number of forms. For example, it can be in written form, as computer files stored on magnetic media, or as displays on a remote terminal. It may be packaged as individual documents, combined into larger documents, or distributed across several documents.*

(2) The Plan for Software Aspects of Certification and the Software Accomplishment Summary may be required as separate printed documents by some certification authorities.

Software life cycle data can be placed in one of two categories associated with the software configuration management controls applied: Control Category 1 (CC1) and Control Category 2 (CC2) (subsection 7.3). This distinction provides a means to manage development costs where less stringent controls can be applied without a reduction in safety. The minimum control category assigned to each data item, and its variation by software level is specified in Annex A.

The following data descriptions define the data generally produced to aid in the software aspects of the certification process. The descriptions are not intended to describe all data necessary to develop a software product, and are not intended to imply a particular data packaging method or organization of the data within a package.

In addition to the data defined in these subsections, other data may be produced as evidence to aid the certification process.

h. Control: If intended to be used for this purpose, this data should be defined in the software plan which defines the process for which the data will be produced. While the nature and

content of this data may vary, as a minimum, CC2 controls should be applied. Examples include memoranda and meeting minutes.

11.1 Plan for Software Aspects of Certification

The Plan for Software Aspects of Certification is the primary means used by the certification authority for determining whether an applicant is proposing a software life cycle that is commensurate with the rigor required for the level of software being developed. This plan should include:

a. System overview: This section provides an overview of the system, including a description of its functions and their allocation to the hardware and software, the architecture, processor(s) used, hardware/software interfaces, and safety features.

b. Software overview: This section briefly describes the software functions with emphasis on the proposed safety and partitioning concepts, for example, resource sharing, redundancy, multiple-version dissimilar software, fault tolerance, and timing and scheduling strategies.

c. Certification considerations: This section provides a summary of the certification basis, including the means of compliance, as relating to the software aspects of certification. This section also states the proposed software level(s) and summarizes the justification provided by the system safety assessment process, including potential software contributions to failure conditions.

d. Software life cycle: This section defines the software life cycle to be used and includes a summary of each software life cycle and its processes for which detailed information is defined in their respective software plans. The summary explains how the objectives of each software life cycle process will be satisfied, and specifies the organizations to be involved, the organizational responsibilities, and the system life cycle processes and certification liaison process responsibilities.

e. Software life cycle data: This section specifies the software life cycle data that will be produced and controlled by the software life cycle processes. This section also describes the relationship of the data to each other or to other data defining the system, the software life cycle data to be submitted to the certification authority, the form of the data, and the means by which software life cycle data will be made available to the certification authority.

f. Schedule: This section describes the means the applicant will use to provide the certification authority with visibility of the activities of the software life cycle processes so reviews can be planned.

g. Additional considerations: This section describes specific features that may affect the certification process, for example, alternative methods of compliance, tool qualification, previously developed software, option-selectable software, user-modifiable software, COTS software, field-loadable software, multiple-version dissimilar software, and product service history.

11.2 Software Development Plan

The Software Development Plan includes the objectives, standards and software life cycle(s) to be used in the software development processes. It may be included in the Plan for Software Aspects of Certification. This plan should include:

a. Standards: Identification of the Software Requirements Standards, Software Design Standards and Software Code Standards for the project. Also, references to the standards for previously developed software, including COTS software, if those standards are different.

b. Software life cycle: A description of the software life cycle processes to be used to form the specific software life cycle(s) to be used on the project, including the transition criteria for

the software development processes. This description is distinct from the summary provided in the Plan for Software Aspects of Certification, in that it provides the detail necessary to ensure proper implementation of the software life cycle processes.

c. Software development environment: A statement of the chosen software development environment in terms of hardware and software, including:

(1) The chosen requirements development method(s) and tools to be used.

(2) The chosen design method(s) and tools to be used.

(3) The programming language(s), coding tools, compilers, linkage editors and loaders to be used.

(4) The hardware platforms for the tools to be used.

11.3 Software Verification Plan

The Software Verification Plan is a description of the verification procedures to satisfy the software verification process objectives. These procedures may vary by software level as defined in the tables of Annex A. This plan should include:

a. Organization: Organizational responsibilities within the software verification process and interfaces with the other software life cycle processes.

b. Independence: A description of the methods for establishing verification independence, when required.

c. Verification methods: A description of the verification methods to be used for each activity of the software verification process.

(1) Review methods, including checklists or other aids.

(2) Analysis methods, including traceability and coverage analysis.

(3) Testing methods, including guidelines that establish the test case selection process, the test procedures to be used, and the test data to be produced.

d. Verification environment: A description of the equipment for testing, the testing and analysis tools, and the guidelines for applying these tools and hardware test equipment (see also paragraph 4.4.3, item b for guidance on indicating target computer and simulator or emulator differences).

e. Transition criteria: The transition criteria for entering the software verification process defined in this plan.

f. Partitioning considerations: If partitioning is used, the methods used to verify the integrity of the partitioning.

g. Compiler assumptions: A description of the assumptions made by the applicant about the correctness of the compiler, linkage editor or loader (paragraph 4.4.2).

h. Reverification guidelines: For software modification, a description of the methods for identifying the affected areas of the software and the changed parts of the Executable Object Code. The reverification should ensure that previously reported errors or classes of errors have been eliminated.

i. Previously developed software: For previously developed software, if the initial compliance baseline for the verification process does not comply with this document, a description of the methods to satisfy the objectives of this document.

j. Multiple-version dissimilar software: If multiple-version dissimilar software is used, a description of the software verification process activities (paragraph 12.3.3).

11.4 Software Configuration Management Plan

The Software Configuration Management Plan establishes the methods to be used to achieve the objectives of the software configuration management (SCM) process throughout the software life cycle. This plan should include:

a. Environment: A description of the SCM environment to be used, including procedures, tools, methods, standards, organizational responsibilities, and interfaces.

b. Activities: A description of the SCM process activities in the software life cycle that will satisfy the objectives for:

(1) Configuration identification: Items to be identified, when they will be identified, the identification methods for software life cycle data (for example, part numbering), and the relationship of software identification and airborne system or equipment identification.

(2) Baselines and traceability: The means of establishing baselines, what baselines will be established, when these baselines will be established, the software library controls, and the configuration item and baseline traceability.

(3) Problem reporting: The content and identification of problem reports for the software product and software life cycle processes, when they will be written, the method of closing problem reports, and the relationship to the change control activity.

(4) Change control: Configuration items and baselines to be controlled, when they will be controlled, the problem/change control activities that control them, pre-certification controls, post-certification controls, and the means of preserving the integrity of baselines and configuration items.

(5) Change review: The method of handling feedback from and to the software life cycle processes; the methods of assessing and prioritizing problems, approving changes, and handling their resolution or change implementation; and the relationship of these methods to the problem reporting and change control activities.

(6) Configuration status accounting: The data to be recorded to enable reporting configuration management status, definition of where that data will be kept, how it will be retrieved for reporting, and when it will be available.

(7) Archive, retrieval, and release: The integrity controls, the release method and authority, and data retention.

(8) Software load control: A description of the software load control safeguards and records.

(9) Software life cycle environment controls: Controls for the tools used to develop, build, verify and load the software, addressing items 1 through 7 above. This includes control of tools to be qualified.

(10) Software life cycle data controls: Controls associated with Control Category 1 and Control Category 2 data.

c. Transition criteria: The transition criteria for entering the SCM process.

d. SCM data: A definition of the software life cycle data produced by the SCM process, including SCM Records, the Software Configuration Index and the Software Life Cycle Environment Configuration Index.

e. Supplier control: The means of applying SCM process requirements to sub-tier suppliers.

11.5 Software Quality Assurance Plan

The Software Quality Assurance Plan establishes the methods to be used to achieve the objectives of the software quality assurance (SQA) process. The SQA Plan may include descriptions of process improvement, metrics, and progressive management methods. This plan should include:

a. Environment: A description of the SQA environment, including scope, organizational responsibilities and interfaces, standards, procedures, tools and methods.

b. Authority: A statement of the SQA authority, responsibility, and independence, including the approval authority for software products.

c. Activities: The SQA activities that are to be performed for each software life cycle process and throughout the software life cycle including:

(1) SQA methods, for example, reviews, audits, reporting, inspections, and monitoring of the software life cycle processes.

(2) Activities related to the problem reporting, tracking and corrective action system.

(3) A description of the software conformity review activity.

d. Transition criteria: The transition criteria for entering the SQA process.

e. Timing: The timing of the SQA process activities in relation to the activities of the software life cycle processes.

f. SQA Records: A definition of the records to be produced by the SQA process.

g. Supplier control: A description of the means of ensuring that sub-tier suppliers' processes and outputs will comply with the SQA Plan.

11.6 Software Requirements Standards

The purpose of Software Requirements Standards is to define the methods, rules and tools to be used to develop the high-level requirements. These standards should include:

a. The methods to be used for developing software requirements, such as structured methods.

b. Notations to be used to express requirements, such as data flow diagrams and formal specification languages.

c. Constraints on the use of the requirement development tools.

d. The method to be used to provide derived requirements to the system process.

11.7 Software Design Standards

The purpose of Software Design Standards is to define the methods, rules and tools to be used to develop the software architecture and low-level requirements. These standards should include:

a. Design description method(s) to be used.

b. Naming conventions to be used.

c. Conditions imposed on permitted design methods, for example, scheduling, and the use of interrupts and event-driven architectures, dynamic tasking, re-entry, global data, and exception handling, and rationale for their use.

d. Constraints on the use of the design tools.

e. Constraints on design, for example, exclusion of recursion, dynamic objects, data aliases, and compacted expressions.

f. Complexity restrictions, for example, maximum level of nested calls or conditional structures, use of unconditional branches, and number of entry/exit points of code components.

11.8 Software Code Standards

The purpose of the Software Code Standards is to define the programming languages, methods, rules and tools to be used to code the software. These standards should include:

a. Programming language(s) to be used and/or defined subset(s). For a programming language, reference the data that unambiguously defines the syntax, the control behavior, the data behavior and side-effects of the language. This may require limiting the use of some features of a language.b. Source Code presentation standards, for example, line length restriction, indentation, and blank line usage and Source Code documentation standards, for example, name of author, revision history, inputs and outputs, and affected global data.

c. Naming conventions for components, subprograms, variables, and constants.

d. Conditions and constraints imposed on permitted coding conventions, such as the degree of coupling between software components and the complexity of logical or numerical expressions and rationale for their use.

e. Constraints on the use of the coding tools.

11.9 Software Requirements Data

Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include:

a. Description of the allocation of system requirements to software, with attention to safety-related requirements and potential failure conditions.

b. Functional and operational requirements under each mode of operation.

c. Performance criteria, for example, precision and accuracy.

d. Timing requirements and constraints.

e. Memory size constraints.

f. Hardware and software interfaces, for example, protocols, formats, frequency of inputs and frequency of outputs.

g. Failure detection and safety monitoring requirements.

h. Partitioning requirements allocated to software, how the partitioned software components interact with each other, and the software level(s) of each partition.

11.10 Design Description

The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the software high-level requirements. This data should include:

a. A detailed description of how the software satisfies the specified software high-level requirements, including algorithms, data structures, and how software requirements are allocated to processors and tasks.

b. The description of the software architecture defining the software structure to implement the requirements.

c. The input/output description, for example, a data dictionary, both internally and externally throughout the software architecture.

d. The data flow and control flow of the design.

e. Resource limitations, the strategy for managing each resource and its limitations, the margins, and the method for measuring those margins, for example, timing and memory.

f. Scheduling procedures and inter-processor/inter-task communication mechanisms, including time-rigid sequencing, preemptive scheduling, Ada rendezvous, and interrupts.

g. Design methods and details for their implementation, for example, software data loading, user-modifiable software, or multiple-version dissimilar software.

h. Partitioning methods and means of preventing partition breaches.

i. Descriptions of the software components, whether they are new or previously developed, and, if previously developed, reference to the baseline from which they were taken.

j. Derived requirements resulting from the software design process.

k. If the system contains deactivated code, a description of the means to ensure that the code cannot be enabled in the target computer.

l. Rationale for those design decisions that are traceable to safety-related system requirements.

11.11 Source Code

This data consists of code written in source language(s) and the compiler instructions for generating the object code from the Source Code, and linking and loading data. This data should include the software identification, including the name and date of revision and/or version, as applicable.

11.12 Executable Object Code

The Executable Object Code consists of a form of Source Code that is directly usable by the central processing unit of the target computer and is, therefore, the software that is loaded into the hardware or system.

11.13 Software Verification Cases and Procedures

Software Verification Cases and Procedures detail how the software verification process activities are implemented. This data should include descriptions of the:

a. Review and analysis procedures: Details, supplementary to the description in the Software Verification Plan, which describes the scope and depth of the review or analysis methods to be used.

b. Test cases: The purpose of each test case, set of inputs, conditions, expected results to achieve the required coverage criteria, and the pass/fail criteria.

c. Test procedures: The step-by-step instructions for how each test case is to be set up and executed, how the test results are evaluated, and the test environment to be used.

11.14 Software Verification Results

The Software Verification Results are produced by the software verification process activities. Software Verification Results should:

a. For each review, analysis and test, indicate each procedure that passed or failed during the activities and the final pass/fail results.

b. Identify the configuration item or software version reviewed, analyzed or tested.

c. Include the results of tests, reviews and analyses, including coverage analyses and traceability analyses.

11.15 Software Life Cycle Environment Configuration Index

The Software Life Cycle Environment Configuration Index (SECI) identifies the configuration of the software life cycle environment. This index is written to aid reproduction of the hardware and software life cycle environment, for software regeneration, reverification, or software modification, and should:

a. Identify the software life cycle environment hardware and its operating system software.

b. Identify the software development tools, such as compilers, linkage editors and loaders, and data integrity tools (such as tools that calculate and embed checksums or cyclical redundancy checks).

c. Identify the test environment used to verify the software product, for example, the software verification tools.

d. Identify qualified tools and their associated tool qualification data.

Note: *This data may be included in the Software Configuration Index.*

11.16 Software Configuration Index

The Software Configuration Index (SCI) identifies the configuration of the software product.

Note: *The SCI can contain one data item or a set (hierarchy) of data items. The SCI can contain the items listed below or it may reference another SCI or other configuration identified data that specifies the individual items and their versions.*

The SCI should identify:

a. The software product.

b. Executable Object Code.

c. Each Source Code component.

d. Previously developed software in the software product, if used.

e. Software life cycle data.

f. Archive and release media.

g. Instructions for building the Executable Object Code, including, for example, instructions and data for compiling and linking; and the procedures used to recover the software for regeneration, testing or modification.

h. Reference to the Software Life Cycle Environment Configuration Index (subsection 11.15), if it is packaged separately.

i. Data integrity checks for the Executable Object Code, if used.

Note: *The SCI may be produced for one software product version or it may be extended to contain data for several alternative or successive software product versions.*

11.17 Problem Reports

Problem reports are a means to identify and record the resolution to software product anomalous behavior, process non-compliance with software plans and standards, and deficiencies in software life cycle data. Problem reports should include:

a. Identification of the configuration item and/or the software life cycle process activity in which the problem was observed.

b. Identification of the configuration item(s) to be modified or a description of the process to be changed.

c. A problem description that enables the problem to be understood and resolved.

d. A description of the corrective action taken to resolve the reported problem.

11.18 Software Configuration Management Records

The results of the SCM process activities are recorded in SCM Records. Examples include configuration identification lists, baseline or software library records, change history reports, archive records, and release records. These examples do not imply records of these specific types need to be produced.

Note: *Due to the integral nature of the SCM process, its outputs will often be included as parts of other software life cycle data.*

11.19 Software Quality Assurance Records

The results of the SQA process activities are recorded in SQA Records. These may include SQA review or audit reports, meeting minutes, records of authorized process deviations, or software conformity review records.

11.20 Software Accomplishment Summary

The Software Accomplishment Summary is the primary data item for showing compliance with the Plan for Software Aspects of Certification. This summary should include:

a. System overview: This section provides an overview of the system, including a description of its functions and their allocation to hardware and software, the architecture, the processor(s) used, the hardware/software interfaces, and safety features. This section also describes any differences from the system overview in the Plan for Software Aspects of Certification.

b. Software overview: This section briefly describes the software functions with emphasis on the safety and partitioning concepts used, and explains differences from the software overview proposed in the Plan for Software Aspects of Certification.

c. Certification considerations: This section restates the certification considerations described in the Plan for Software Aspects of Certification and describes any differences.

d. Software characteristics: This section states the Executable Object Code size, timing and memory margins, resource limitations, and the means of measuring each characteristic.

e. Software life cycle: This section summarizes the actual software life cycle(s) and explains differences from the software life cycle and software life cycle processes proposed in the Plan for Software Aspects of Certification.

f. Software life cycle data: This section references the software life cycle data produced by the software development processes and integral processes. It describes the relationship of the data to each other and to other data defining the system, and the means by which

software life cycle data will be made available to the certification authority. This section also describes any differences from the Plan for Software Aspects of Certification.

g. Additional considerations: This section summarizes certification issues that may warrant the attention of the certification authority and references data items applicable to these issues, such as issue papers or special conditions.

h. Software identification: This section identifies the software configuration by part number and version.

i. Change history: If applicable, this section includes a summary of software changes with attention to changes made due to failures affecting safety, and identifies changes from the software life cycle processes since the previous certification.

j. Software status: This section contains a summary of problem reports unresolved at the time of certification, including a statement of functional limitations.

k. Compliance statement: This section includes a statement of compliance with this document and a summary of the methods used to demonstrate compliance with criteria specified in the software plans. This section also addresses additional rulings and deviations from the software plans, standards and this document.

12.0 ADDITIONAL CONSIDERATIONS

The previous sections of this document provide guidance for satisfying certification requirements in which the applicant submits evidence of the software life cycle processes. This section introduces additional considerations concerning the software aspects of certification in relation to the use of previously developed software, qualification of software tools, and alternative methods for achieving the objectives of the previous sections of this document.

12.1 Use of Previously Developed Software

The guidelines of this subsection discuss the issues associated with the use of previously developed software, including the assessment of modifications, the effect of changing an aircraft installation, application environment, or development environment, upgrading a development baseline, and SCM and SQA considerations. The intention to use previously developed software is stated in the Plan for Software Aspects of Certification.

12.1.1 Modifications to Previously Developed Software

This guidance discusses modifications to previously developed software where the outputs of the previous software life cycle processes comply with this document. Modification may result from requirement changes, the detection of errors, and/or software enhancements. Analysis activities for proposed modifications include:

a. The revised outputs of the system safety assessment process should be reviewed considering the proposed modifications.

b. If the software level is revised, the guidelines of paragraph 12.1.4 should be considered.

c. Both the impact of the software requirements changes and the impact of software architecture changes should be analyzed, including the consequences of software requirement changes upon other requirements and coupling between several software components that may result in reverification effort involving more than the modified area.

d. The area affected by a change should be determined. This may be done by data flow analysis, control flow analysis, timing analysis and traceability analysis.

e. Areas affected by the change should be reverified considering the guidelines of section 6.

12.1.2 Change of Aircraft Installation

Airborne systems or equipment containing software which has been previously certified at a certain software level and under a specific certification basis may be used in a new aircraft installation. This guidance should be used for new aircraft installations, if using previously developed software:

a. The system safety assessment process assesses the new aircraft installation and determines the software level and the certification basis. No additional effort will be required if these are the same for the new installation as they were in the previous installation.

b. If functional modifications are required for the new installation, the guidelines of paragraph 12.1.1, Modifications to Previously Developed Software, should be satisfied.

c. If the previous development activity did not produce outputs required to substantiate the safety objectives of the new installation, the guidelines of paragraph 12.1.4, Upgrading A Development Baseline, should be satisfied.

12.1.3 Change of Application or Development Environment

Use and modification of previously developed software may involve a new development environment, a new target processor or other hardware, or integration with other software than that used for the original application.

New development environments may increase or reduce some activities within the software life cycle. New application environments may require activities in addition to software life cycle process activities which address modifications. Guidance for change of application or development environment includes:

a. If a new development environment uses software development tools, the guidelines of subsection 12.2, Tool Qualification, may be applicable.

b. The rigor of the evaluation of an application change should consider the complexity and sophistication of the programming language. For example, the rigor of the evaluation for Ada generics will be greater if the generic parameters are different in the new application. For object oriented languages, the rigor will be greater if the objects that are inherited are different in the new application.

c. If a different compiler or different set of compiler options are used, resulting in different object code, the results from a previous software verification process activity using the object code may not be valid and should not be used for the new application. In this case, previous test results may no longer be valid for the structural coverage criteria of the new application. Similarly, compiler assumptions about optimization may not be valid.

d. If a different processor is used then:

   (1) The results from a previous software verification process activity directed at the hardware/software interface should not be used for the new application.

   (2) The previous hardware/software integration tests should be executed for the new application.

   (3) Reviews of hardware/software compatibility should be repeated.

   (4) Additional hardware/software integration tests and reviews may be necessary.

e. Verification of software interfaces should be conducted where previously developed software is used with different interfacing software.

12.1.4 Upgrading A Development Baseline

Guidelines follow for software whose software life cycle data from a previous application are determined to be inadequate or do not satisfy the objectives of this document, due to the safety objectives associated with a new application. These guidelines are intended to aid in the acceptance of:

- Commercial off-the-shelf software.
- Airborne software developed to other guidelines.
- Airborne software developed prior to the existence of this document.
- Software previously developed to this document at a lower software level.

Guidance for upgrading a development baseline includes:

a. The objectives of this document should be satisfied while taking advantage of software life cycle data of the previous development that satisfy the objectives for the new application.

b. Software aspects of certification should be based on the failure conditions and software level(s) as determined by the system safety assessment process. Comparison to failure conditions of the previous application will determine areas which may need to be upgraded.

c. Software life cycle data from a previous development should be evaluated to ensure that the software verification process objectives of the software level are satisfied for the new application.

d. Reverse engineering may be used to regenerate software life cycle data that is inadequate or missing in satisfying the objectives of this document. In addition to producing the software product, additional activities may need to be performed to satisfy the software verification process objectives.

e. If use of product service history is planned to satisfy the objectives of this document in upgrading a development baseline, the guidelines of paragraph 12.3.5 should be considered.

f. The applicant should specify the strategy for accomplishing compliance with this document in the Plan for Software Aspects of Certification.

12.1.5 Software Configuration Management Considerations

If previously developed software is used, the software configuration management process for the new application should include, in addition to the guidelines of section 7:

a. Traceability from the software product and software life cycle data of the previous application to the new application.

b. Change control that enables problem reporting, problem resolution, and tracking of changes to software components used in more than one application.

12.1.6 Software Quality Assurance Considerations

If previously developed software is used, the software quality assurance process for the new application should include, in addition to the guidelines of section 8:

a. Assurance that the software components satisfy or exceed the software life cycle criteria of the software level for the new application.

b. Assurance that changes to the software life cycle processes are stated in the software plans.

12.2 Tool Qualification

Qualification of a tool is needed when processes of this document are eliminated, reduced or automated by the use of a software tool without its output being verified as specified in section 6. The use of software tools to automate activities of the software life cycle processes can help satisfy system safety objectives insofar as they can enforce conformance with software development standards and use automatic checks.

The objective of the tool qualification process is to ensure that the tool provides confidence at least equivalent to that of the process(es) eliminated, reduced or automated. If partitioning of tool functions can be demonstrated, only those functions that are used to eliminate, reduce, or automate software life cycle process activities, and whose outputs are not verified, need be qualified.

Only deterministic tools may be qualified, that is, tools which produce the same output for the same input data when operating in the same environment. The tool qualification process may be applied either to a single tool or to a collection of tools.

Software tools can be classified as one of two types:

- Software development tools: Tools whose output is part of airborne software and thus can introduce errors. For example, a tool which generates Source Code directly from low-level requirements would have to be qualified if the generated Source Code is not verified as specified in section 6.
- Software verification tools: Tools that cannot introduce errors, but may fail to detect them. For example, a static analyzer, that automates a software verification process activity,

should be qualified if the function that it performs is not verified by another activity. Type checkers, analysis tools and test tools are other examples.

Tool qualification guidance includes:

a. Tools should be qualified according to the type specified above.

b. Combined software development tools and software verification tools should be qualified to comply with the guidelines in paragraph 12.2.1, unless partitioning between the two functions can be demonstrated.

c. The software configuration management process and software quality assurance process objectives for airborne software should apply to software tools to be qualified.

The software verification process objectives for software development tools are described in paragraph 12.2.1, item d.

A tool may be qualified only for use on a specific system where the intention to use the tool is stated in the Plan for Software Aspects of Certification. Use of the tool for other systems may need further qualification.

12.2.1 Qualification Criteria for Software Development Tools

The qualification criteria for software development tools includes:

a. If a software development tool is to be qualified, the software development processes for the tool should satisfy the same objectives as the software development processes of airborne software.

b. The software level assigned to the tool should be the same as that for the airborne software it produces, unless the applicant can justify a reduction in software level of the tool to the certification authority.

Note: *A reduction in a tool's software level can be based upon the significance of the software verification process activity to be eliminated, reduced or automated, with respect to the entire suite of verification activities. This significance is a function of:*

- *The type of software verification process activity to be eliminated, reduced or automated. For example, a verification activity for conformance of the Source Code with software indentation standards is less significant than verification activity for compliance of the Executable Object Code with the high-level requirements.*
- *The likelihood that other verification activities would have detected the same error(s).*

c. The applicant should demonstrate that the tool complies with its Tool Operational Requirements (subparagraph 12.2.3.2). This demonstration may involve a trial period during which a verification of the tool output is performed and tool-related problems are analyzed, recorded and corrected.

d. Software development tools should be verified to check the correctness, consistency, and completeness of the Tool Operational Requirements and to verify the tool against those requirements. The objectives of the tool's software verification process are different from those of the airborne software since the tool's high-level requirements correspond to its Tool Operational Requirements instead of system requirements. Verification of software development tools may be achieved by:

(1) Review of the Tool Operational Requirements as described in paragraph 6.3.1, items a and b.

(2) Demonstration that the tool complies with its Tool Operational Requirements under normal operating conditions.

(3) Demonstration that the tool complies with its Tool Operational Requirements while executing in abnormal operating conditions, including external disturbances and selected failures applied to the tool and its environment.

(4) Requirements-based coverage analysis and additional tests to complete the coverage of the requirements.

(5) Structural coverage analysis appropriate for the tool's software level.

(6) Robustness testing for tools with a complex data flow or control flow, as specified in subparagraph 6.4.2.2, appropriate to the tool's software level.

(7) Analysis of potential errors produced by the tool, to confirm the validity of the Tool Qualification Plan.

12.2.2 Qualification Criteria for Software Verification Tools

The qualification criteria for software verification tools should be achieved by demonstration that the tool complies with its Tool Operational Requirements under normal operational conditions.

12.2.3 Tool Qualification Data

Guidance for tool qualification data includes:

a. When qualifying a tool, the Plan for Software Aspects of Certification of the related airborne software should specify the tool to be qualified and reference the tool qualification data.

b. The tool qualification data should be controlled as Control Category 1 (CC1) for software development tools and CC2 for software verification tools.

c. For software development tools, the tool qualification data should be consistent with the data in section 11 and have the same characteristics and content as data for airborne software, with these considerations:

(1) A Tool Qualification Plan satisfies the same objectives as the Plan for Software Aspects of Certification of the airborne software.

(2) Tool Operational Requirements satisfies the same objectives as the Software Requirements Data of the airborne software.

(3) A Tool Accomplishment Summary satisfies the same objectives as the Software Accomplishment Summary of the airborne software.

12.2.3.1 Tool Qualification Plan

For software development tools to be qualified, the Tool Qualification Plan describes the tool qualification process. This plan should include:

a. Configuration identification of the tool.

b. Details of the certification credit sought, that is, the software verification process activities to be eliminated, reduced or automated.

c. The software level proposed for the tool.

d. A description of the tool's architecture.

e. The tool qualification activities to be performed.

f. The tool qualification data to be produced.

12.2.3.2 Tool Operational Requirements

Tool Operational Requirements describe the tool's operational functionality. This data should include:

a. A description of the tool's functions and technical features. For software development tools, it includes the software development process activities performed by the tool.

b. User information, such as installation guides and user manuals.

c. A description of the tool's operational environment.

d. For software development tools, the expected responses of the tool under abnormal operating conditions.

12.2.4 Tool Qualification Agreement

The certification authority gives its agreement to the use of a tool in two steps:

- For software development tools, agreement with the Tool Qualification Plan. For software verification tools, agreement with the Plan for Software Aspects of Certification of the airborne software.
- For software development tools, agreement with the Tool Accomplishment Summary. For software verification tools, agreement with the Software Accomplishment Summary of the airborne software.

12.3 Alternative Methods

Some methods were not discussed in the previous sections of this document because of inadequate maturity at the time this document was written or limited applicability for airborne software. It is not the intention of this document to restrict the implementation of any current or future methods. Any single alternative method discussed in this subsection is not considered an alternative to the set of methods recommended by this document, but may be used in satisfying one or more of the objectives of in this document.

Alternative methods may be used to support one another. For example, formal methods may assist tool qualification or a qualified tool may assist the use of formal methods.

An alternative method cannot be considered in isolation from the suite of software development processes. The effort for obtaining certification credit of an alternative method is dependent on the software level and the impact of the alternative method on the software life cycle processes. Guidance for using an alternative method includes:

a. An alternative method should be shown to satisfy the objectives of this document.

b. The applicant should specify in the Plan for Software Aspects of Certification, and obtain agreement from the certification authority for:

   (1) The impact of the proposed method on the software development processes.

   (2) The impact of the proposed method on the software life cycle data.

   (3) The rationale for use of the alternative method which shows that the system safety objectives are satisfied.

c. The rationale should be substantiated by software plans, processes, expected results, and evidence of the use of the method.

12.3.1 Formal Methods

Formal methods involve the use of formal logic, discrete mathematics, and computer-readable languages to improve the specification and verification of software. These methods could produce an implementation whose operational behavior is known with confidence to be within a defined domain. In their most thorough application, formal methods could be equivalent to exhaustive analysis of a system with respect to its requirements. Such analysis could provide:

- Evidence that the system is complete and correct with respect to its requirements.
- Determination of which code, software requirements or software architecture satisfy the next higher level of software requirements.

The goal of applying formal methods is to prevent and eliminate requirements, design and code errors throughout the software development processes. Thus, formal methods are complementary to testing. Testing shows that functional requirements are satisfied and detects errors, and formal methods could be used to increase confidence that anomalous behavior will not occur (for inputs that are out of range) or unlikely to occur.

Formal methods may be applied to software development processes with consideration of these factors:

- Levels of the design refinement: The use of formal methods begins by specifying software high-level requirements in a formal specification language and verifying by formal proofs that they satisfy system requirements, especially constraints on acceptable operation. The next lower level of requirements are then shown to satisfy the high-level requirements. Performing this process down to the Source Code provides evidence that the software satisfies system requirements. Application of formal methods can start and stop with consecutive levels of the design refinement, providing evidence that those levels of requirements are specified correctly.
- Coverage of software requirements and software architecture: Formal methods may be applied to software requirements that:
  - Are safety-related.
  - Can be defined by discrete mathematics.
  - Involve complex behavior, such as concurrency, distributed processing, redundancy management, and synchronization.

  These criteria can be used to determine the set of requirements at the level of the design refinement to which the formal methods are applied.
- Degree of rigor: Formal methods include these increasingly rigorous levels:
  - Formal specification with no proofs.
  - Formal specification with manual proofs.
  - Formal specification with automatically checked or generated proofs.

  The use of formal specifications alone forces requirements to be unambiguous. Manual proof is a well-understood process that can be used when there is little detail. Automatically checked or generated proofs can aid the human proof process and offer a higher degree of dependability, especially for more complicated proofs.

12.3.2 Exhaustive Input Testing

There are situations where the software component of an airborne system or equipment is simple and isolated such that the set of inputs and outputs can be bounded. If so, it may be possible to demonstrate that exhaustive testing of this input space can be substituted for a software verification process activity. For this alternative method, the applicant should include:

a. A complete definition of the set of valid inputs and outputs of the software.

b. An analysis which confirms the isolation of the inputs to the software.

c. Rationale for the exhaustive input test cases and procedures.

d. The test cases, test procedures and test results.

12.3.3 Considerations for Multiple-Version Dissimilar Software Verification

Guidelines follow concerning the software verification process as it applies to multiple-version dissimilar software. If the software verification process is modified because of the use of multiple-version dissimilar software, evidence should be provided that the software verification process objectives are satisfied and that equivalent error detection is achieved for each software version.

Multiple, dissimilar versions of the software are produced using combinations of these techniques:

- The Source Code is implemented in two or more different programming languages.
- The object code is generated using two or more different compilers.
- Each software version of Executable Object Code executes on a separate, dissimilar processor, or on a single processor with the means to provide partitioning between the software versions.
- The software requirements, software design, and/or Source Code are developed by two or more development teams whose interactions are managed.
- The software requirements, software design, and/or Source Code are developed on two or more software development environments, and/or each version is verified using separate test environments.
- The Executable Object Code is linked and loaded using two or more different linkage editors and two or more different loaders.
- The software requirements, software design, and/or Source Code are developed in conformance with two or more different Software Requirements Standards, Software Design Standards, and/or Software Code Standards, respectively.

When multiple versions of software are used, the software verification methods may be modified from those used to verify single version software. They will apply to software development process activities that are multi-thread, such as separate, multiple development teams. The software verification process is dependent on the combined hardware and software architectures since this affects the dissimilarity of the multiple software versions. Additional software verification process objectives to be satisfied are:

a. To demonstrate that the inter-version compatibility requirements are satisfied, including compatibility during normal and abnormal operations and state transitions.

b. To demonstrate that equivalent error detection is achieved.

Other changes in software verification process activities may be agreed with by the certification authority, if the changes are substantiated by rationale that confirms equivalent software verification coverage.

12.3.3.1 Independence of Multiple-Version Dissimilar Software

When multiple-version dissimilar software versions are developed independently using a managed method, the development processes have the potential to reveal certain classes of errors such that verification of each software version is equivalent to independent verification of the software development processes. To realize the advantage of this potential, guidance for independence includes:

a. The applicant should demonstrate that different teams with limited interaction developed each software version's software requirements, software design and Source Code.

b. Independent test coverage analyses should still be performed as with a single version.

12.3.3.2 Multiple Processor-Related Verification

When each version of dissimilar software executes on a different type of processor, the verification of some aspects of compatibility of the code with the processor (paragraph 6.4.3, item a) may be replaced by verification to ensure that the multiple types of processor produce the correct outputs. This verification consists of integration tests in which the outputs of the multiple versions are cross-compared in requirements-based test cases. The applicant should show that:

a. Equivalent error detection is achieved.

b. Each processor was designed by a different developer.

c. The outputs of the multiple versions are equivalent.

12.3.3.3 Multiple-Version Source Code Verification

The guidelines for structural coverage analysis (subparagraph 6.4.4.2) may be modified for airborne systems or equipment using multiple-version dissimilar software. Structural coverage analysis may be performed at the Source Code level even if the object code is not directly traceable to Source Code statements provided that the applicant shows that:

a. Each version of software is coded using a different programming language.

b. Each compiler used is from a different developer.

12.3.3.4 Tool Qualification for Multiple-Version Dissimilar Software

If multiple-version dissimilar software is used, the tool qualification process may be modified, if evidence is available that the multiple software development tools are dissimilar. This depends on the demonstration of equivalent software verification process activity in the development of the multiple software versions using dissimilar software development tools. The applicant should show that:

a. Each tool was obtained from a different developer.

b. Each tool has a dissimilar design.

12.3.3.5 Multiple Simulators and Verification

If separate, dissimilar simulators are used to verify multiple-version dissimilar software versions, then tool qualification of the simulators may be modified. This depends on the demonstration of equivalent software verification process activity in the simulation of the multiple software versions using multiple simulators. Unless it can be justified as unnecessary, for multiple simulators to be dissimilar, evidence should be available that:

a. Each simulator was developed by a different team.

b. Each simulator has different requirements, a different design and a different programming language.

c. Each simulator executes on a different processor.

Note: *When a multiple processor system using multiple, dissimilar versions of software are executing on identical processors, it may be difficult to demonstrate dissimilarity of simulators because of the reliance on information obtained from a common source, the processor manufacturer.*

12.3.4 Software Reliability Models

During the preparation of this document, methods for estimating the post-verification probabilities of software errors were examined. The goal was to develop numerical requirements for such probabilities for software in computer-based airborne systems or equipment. The conclusion reached, however, was that currently available methods do not provide results in which confidence can be placed to the level required for this purpose. Hence, this document does not provide guidance for software error rates. If the applicant proposes to use software reliability models for certification credit, rationale for the model should be included in the Plan for Software Aspects of Certification, and agreed with by the certification authority.

12.3.5 Product Service History

If equivalent safety for the software can be demonstrated by the use of the software's product service history, some certification credit may be granted. The acceptability of this method is dependent on:

- Configuration management of the software.
- Effectiveness of problem reporting activity.
- Stability and maturity of the software.
- Relevance of product service history environment.
- Actual error rates and product service history.
- Impact of modifications.

Guidance for the use of product service history includes:

a. The applicant should show that the software and associated evidence used to comply with system safety objectives have been under configuration management throughout the product service history.

b. The applicant should show that the problem reporting during the product service history provides assurance that representative data is available and that in-service problems were reported and recorded, and are retrievable.

c. Configuration changes during the product service history should be identified and the effect analyzed to confirm the stability and maturity of the software. Uncontrolled changes to the Executable Object Code during the product service history may invalidate the use of product service history.

d. The intended software usage should be analyzed to show the relevance of the product service history.

e. If the operating environments of the existing and proposed applications differ, additional software verification should confirm compliance with the system safety objectives.

f. The analysis of configuration changes and product service history environment may require the use of software requirements and design data to confirm the applicability of the product service history environment.

g. If the software is a subset of the software that was active during the service period, then analysis should confirm the equivalency of the new environment with the previous environment, and determine those software components that were not executed during normal operation.

Note: *Additional verification may be needed to confirm compliance with the system safety objectives for those components.*

h. The problem report history should be analyzed to determine how safety-related problems occurred and which problems were corrected.

i. Those problems that are indicative of an inadequate process, such as design or code errors, should be indicated separately from those whose cause are outside the scope of this document, such as hardware or system requirements errors.

j. The data described above and these items should be specified in the Plan for Software Aspects of Certification:

(1) Analysis of the relevance of the product service history environment.

(2) Length of service period and rationale for calculating the number of hours in service, including factors such as operational modes, the number of independently operating copies in the installation and in service, and the definition of "normal operation" and "normal operation time."

(3) Definition of what was counted as an error and rationale for that definition.

(4) Proposed acceptable error rates and rationale for the product service history period in relation to the system safety and proposed error rates.

k. If the error rate is greater than that identified in the plan, these errors should be analyzed and the analyses reviewed with the certification authority.

ANNEX A PROCESS OBJECTIVES AND OUTPUTS BY SOFTWARE LEVEL

This annex provides guidelines for the software life cycle process objectives and outputs described in this document by software level. These tables reference the objectives and outputs of the software life cycle processes previously described in this document.

The tables include guidelines for:

a. The process objectives applicable for each software level. For level E software, see paragraph 2.2.2.

b. The independence by software level of the software life cycle process activities applicable to satisfy that process's objectives.

c. The control category by software level for the software life cycle data produced by the software life cycle process activities (subsection 7.3).

Table A-1

Software Planning Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Software development and integral processes activities are defined.	4.1a 4.3	○	○	○	○	Plan for Software Aspects of Certification	11.1	①	①	①	①
							Software Development Plan	11.2	①	①	②	②
							Software Verification Plan	11.3	①	①	②	②
							SCM Plan	11.4	①	①	②	②
							SQA Plan	11.5	①	①	②	②
2	Transition criteria, inter-relationships and sequencing among processes are defined.	4.1b 4.3	○	○	○							
3	Software life cycle environment is defined.	4.1c	○	○	○							
4	Additional considerations are addressed.	4.1d	○	○	○	○						
5	Software development standards are defined.	4.1e	○	○	○		SW Requirements Standards	11.6	①	①	②	
							SW Design Standards	11.7	①	①	②	
							SW Code Standards	11.8	①	①	②	
6	Software plans comply with this document.	4.1f 4.6	○	○	○		SQA Records	11.19	②	②	②	
							Software Verification Results	11.14	②	②	②	
7	Software plans are coordinated.	4.1g 4.6	○	○	○		SQA Records	11.19	②	②	②	
							Software Verification Results	11.14	②	②	②	

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-2

Software Development Processes

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	High-level requirements are developed.	5.1.1a	○	○	○	○	Software Requirements Data	11.9	①	①	①	①
2	Derived high-level requirements are defined.	5.1.1b	○	○	○	○	Software Requirements Data	11.9	①	①	①	①
3	Software architecture is developed.	5.2.1a	○	○	○	○	Design Description	11.10	①	①	②	②
4	Low-level requirements are developed.	5.2.1a	○	○	○	○	Design Description	11.10	①	①	②	②
5	Derived low-level requirements are defined.	5.2.1b	○	○	○	○	Design Description	11.10	①	①	②	②
6	Source Code is developed.	5.3.1a	○	○	○	○	Source Code	11.11	①	①	①	①
7	Executable Object Code is produced and integrated in the target computer.	5.4.1a	○	○	○	○	Executable Object Code	11.12	①	①	①	①

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

76

Table A-3

Verification Of Outputs of Software Requirements Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Software high-level requirements comply with system requirements.	6.3.1a	●	●	○	○	Software Verification Results	11.14	②	②	②	②
2	High-level requirements are accurate and consistent.	6.3.1b	●	●	○	○	Software Verification Results	11.14	②	②	②	②
3	High-level requirements are compatible with target computer.	6.3.1c	○	○			Software Verification Results	11.14	②	②		
4	High-level requirements are verifiable.	6.3.1d	○	○	○		Software Verification Results	11.14	②	②	②	
5	High-level requirements conform to standards.	6.3.1e	○	○	○		Software Verification Results	11.14	②	②	②	
6	High-level requirements are traceable to system requirements.	6.3.1f	○	○	○	○	Software Verification Results	11.14	②	②	②	②
7	Algorithms are accurate.	6.3.1g	●	●	○		Software Verification Results	11.14	②	②	②	

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-4

Verification Of Outputs of Software Design Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Low-level requirements comply with high-level requirements.	6.3.2a	●	●	○		Software Verification Results	11.14	②	②	②	
2	Low-level requirements are accurate and consistent.	6.3.2b	●	●	○		Software Verification Results	11.14	②	②	②	
3	Low-level requirements are compatible with target computer.	6.3.2c	○	○			Software Verification Results	11.14	②	②		
4	Low-level requirements are verifiable.	6.3.2d	○	○			Software Verification Results	11.14	②	②		
5	Low-level requirements conform to standards.	6.3.2e	○	○	○		Software Verification Results	11.14	②	②	②	
6	Low-level requirements are traceable to high-level requirements.	6.3.2f	○	○	○		Software Verification Results	11.14	②	②	②	
7	Algorithms are accurate.	6.3.2g	●	●	○		Software Verification Results	11.14	②	②	②	
8	Software architecture is compatible with high-level requirements.	6.3.3a	●	○	○		Software Verification Results	11.14	②	②	②	
9	Software architecture is consistent.	6.3.2b	●	○	○		Software Verification Results	11.14	②	②	②	
10	Software architecture is compatible with target computer.	6.3.3c	○	○			Software Verification Results	11.14	②	②		
11	Software architecture is verifiable.	6.3.3d	○	○			Software Verification Results	11.14	②	②		
12	Software architecture conforms to standards.	6.3.3e	○	○	○		Software Verification Results	11.14	②	②	②	
13	Software partitioning integrity is confirmed.	6.3.3f	●	○	○	○	Software Verification Results	11.14	②	②	②	②

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

78

Table A-5

Verification Of Outputs of Software Coding & Integration Processes

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Source Code complies with low-level requirements.	6.3.4a	●	●	○		Software Verification Results	11.14	②	②	②	
2	Source Code complies with software architecture.	6.3.4b	●	○	○		Software Verification Results	11.14	②	②	②	
3	Source Code is verifiable.	6.3.4c	○	○			Software Verification Results	11.14	②	②		
4	Source Code conforms to standards.	6.3.4d	○	○	○		Software Verification Results	11.14	②	②	②	
5	Source Code is traceable to low-level requirements.	6.3.4e	○	○	○		Software Verification Results	11.14	②	②	②	
6	Source Code is accurate and consistent.	6.3.4f	●	○	○		Software Verification Results	11.14	②	②	②	
7	Output of software integration process is complete and correct.	6.3.5	○	○	○		Software Verification Results	11.14	②	②	②	

LEGEND:	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-6

Testing Of Outputs of Integration Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Executable Object Code complies with high-level requirements.	6.4.2.1 6.4.3	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
2	Executable Object Code is robust with high-level requirements.	6.4.2.2 6.4.3	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
3	Executable Object Code complies with low-level requirements.	6.4.2.1 6.4.3	●	●	○		Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	
4	Executable Object Code is robust with low-level requirements.	6.4.2.2 6.4.3	●	○	○		Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	
5	Executable Object Code is compatible with target computer.	6.4.3a	○	○	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-7

Verification Of Verification Process Results

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Test procedures are correct.	6.3.6b	●	○	○		Software Verification Cases and Procedures	11.13	②	②	②	
2	Test results are correct and discrepancies explained.	6.3.6c	●	○	○		Software Verification Results	11.14	②	②	②	
3	Test coverage of high-level requirements is achieved.	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision) is achieved.	6.4.4.2	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●			Software Verification Results	11.14	②	②		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.2a 6.4.4.2b	●	●	○		Software Verification Results	11.14	②	②	②	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.2c	●	●	○		Software Verification Results	11.14	②	②	②	

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-8

Software Configuration Management Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Configuration items are identified.	7.2.1	○	○	○	○	SCM Records	11.18	②	②	②	②
2	Baselines and traceability are established.	7.2.2	○	○	○	○	Software Configuration Index	11.16	①	①	①	①
							SCM Records	11.18	②	②	②	②
3	Problem reporting, change control, change review, and configuration status accounting are established.	7.2.3 7.2.4 7.2.5 7.2.6	○	○	○	○	Problem Reports	11.17	②	②	②	②
							SCM Records	11.18	②	②	②	②
4	Archive, retrieval, and release are established.	7.2.7	○	○	○	○	SCM Records	11.18	②	②	②	②
5	Software load control is established.	7.2.8	○	○	○	○	SCM Records	11.18	②	②	②	②
6	Software life cycle environment control is established.	7.2.9	○	○	○	○	Software Life Cycle Environment Configuration Index	11.15	①	①	①	②
							SCM Records	11.18	②	②	②	②

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Note: *(1) Although the software configuration management objectives of section 7 do not vary with software level, the control category assigned to the software life cycle data may vary.*

(2) The objectives of section 7 provide a sufficient integrity basis in the SCM process activities without the need for the independence criteria.

Table A-9

Software Quality Assurance Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Assurance is obtained that software development and integral processes comply with approved software plans and standards.	8.1a	●	●	●	●	Software Quality Assurance (SQA) Records	11.19	②	②	②	②
2	Assurance is obtained that transition criteria for the software life cycle processes are satisfied.	8.1b	●	●			SQA Records	11.19	②	②		
3	Software conformity review is conducted.	8.1c 8.3	●	●	●	●	SQA Records	11.19	②	②	②	②

LEGEND:

●	The objective should be satisfied with independence.
○	The objective should be satisfied.
Blank	Satisfaction of objective is at applicant's discretion.
①	Data satisfies the objectives of Control Category 1 (CC1).
②	Data satisfies the objectives of Control Category 2 (CC2).

Table A-10

Certification Liaison Process

	Objective		Applicability by SW Level				Output		Control Category by SW level			
	Description	Ref.	A	B	C	D	Description	Ref.	A	B	C	D
1	Communication and understanding between the applicant and the certification authority is established.	9.0	○	○	○	○	Plan for Software Aspects of Certification	11.1	①	①	①	①
2	The means of compliance is proposed and agreement with the Plan for Software Aspects of Certification is obtained.	9.1	○	○	○	○	Plan for Software Aspects of Certification	11.1	①	①	①	①
3	Compliance substantiation is provided.	9.2	○	○	○	○	Software Accomplishment Summary	11.20	①	①	①	①
							Software Configuration Index	11.16	①	①	①	①

LEGEND:		
	●	The objective should be satisfied with independence.
	○	The objective should be satisfied.
	Blank	Satisfaction of objective is at applicant's discretion.
	①	Data satisfies the objectives of Control Category 1 (CC1).
	②	Data satisfies the objectives of Control Category 2 (CC2).

Note: *The Plan for Software Aspects of Certification and the Software Configuration Index are outputs of other processes. They are included in this table to show completion of the certification liaison process objectives.*

THIS PAGE INTENTIONALLY LEFT BLANK

ANNEX BACRONYMS AND GLOSSARY OF TERMS

Acronyms

AC	Advisory Circular
AMJ	Advisory Material - Joint
CC1	Control Category 1
CC2	Control Category 2
COTS	commercial off-the-shelf
EUROCAE	European Organisation for Civil Aviation Equipment
FAA	Federal Aviation Administration
FAR	Federal Aviation Regulation
IC	integrated circuit
I/O	input and/or output
JAA	Joint Aviation Authorities
JAR	Joint Aviation Requirements
RTCA	RTCA, Inc.
SCI	Software Configuration Index
SCM	software configuration management
SECI	Software Life Cycle Environment Configuration Index
SQA	software quality assurance

Glossary

These definitions are provided for the terms which are used in this document. If a term is not defined in this annex, it is possible that it is defined instead in the body of this document. Refer to the American Heritage Dictionary for the definition of common terms.

Algorithm - A finite set of well-defined rules that gives a sequence of operations for performing a specific task.

Anomalous behavior - Behavior that is inconsistent with specified requirements.

Applicant - A person or organization seeking approval from the certification authority.

Approval - The act or instance of expressing a favorable opinion or giving formal or official sanction.

Assurance - The planned and systematic actions necessary to provide adequate confidence and evidence that a product or process satisfies given requirements.

Audit - An independent examination of the software life cycle processes and their outputs to confirm required attributes.

Baseline - The approved, recorded configuration of one or more configuration items, that thereafter serves as the basis for further development, and that is changed only through change control procedures.

Certification - Legal recognition by the certification authority that a product, service, organization or person complies with the requirements. Such certification comprises the activity of technically checking the product, service, organization or person and the formal recognition of compliance with the applicable requirements by issue of a certificate, license, approval or other documents as required by national laws and procedures. In particular, certification of a product involves: (a) the process of assessing the design of a product to ensure that it complies with a set of standards applicable to that type of product so as to demonstrate an acceptable level of safety; (b) the process of assessing an individual product to ensure that it conforms with the certified type design; (c) the issuance of a certificate required by national laws to declare that compliance or conformity has been found with standards in accordance with items (a) or (b) above.

Certification Authority - The organization or person responsible within the state or country concerned with the certification of compliance with the requirements.

Note: *A matter concerned with aircraft, engine or propeller type certification or with equipment approval would usually be addressed by the certification authority; matters concerned with continuing airworthiness might be addressed by what would be referred to as the airworthiness authority.*

Certification credit - Acceptance by the certification authority that a process, product or demonstration satisfies a certification requirement.

Change control - (1) The process of recording, evaluating, approving or disapproving and coordinating changes to configuration items after formal establishment of their configuration identification or to baselines after their establishment. (2) The systematic evaluation, coordination, approval or disapproval and implementation of approved changes in the configuration of a configuration item after formal establishment of its configuration identification or to baselines after their establishment.

Note: *This term may be called configuration control in other industry standards.*

Code - The implementation of particular data or a particular computer program in a symbolic form, such as source code, object code or machine code.

Commercial off-the-shelf (COTS) software - Commercially available applications sold by vendors through public catalog listings. COTS software is not intended to be customized or enhanced. Contract-negotiated software developed for a specific application is not COTS software.

Compiler - Program that translates source code statements of a high level language, such as FORTRAN or Pascal, into object code.

Component - A self-contained part, combination of parts, sub-assemblies or units, which performs a distinct function of a system.

Condition - A Boolean expression containing no Boolean operators.

Condition/Decision Coverage - Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken on all possible outcomes at least once, and every decision in the program has taken on all possible outcomes at least once.

Configuration identification - (1) The process of designating the configuration items in a system and recording their characteristics. (2) The approved documentation that defines a configuration item.

Configuration item - (1) One or more hardware or software components treated as a unit for configuration management purposes. (2) Software life cycle data treated as a unit for configuration management purposes.

Configuration management - (1) The process of identifying and defining the configuration items of a system, controlling the release and change of these items throughout the software life cycle, recording and reporting the status of configuration items and change requests and verifying the completeness and correctness of configuration items. (2) A discipline applying technical and administrative direction and surveillance to (a) identify and record the functional and physical characteristics of a configuration item, (b) control changes to those characteristics, and (c) record and report change control processing and implementation status.

Configuration status accounting - The recording and reporting of the information necessary to manage a configuration effectively, including a listing of the approved configuration identification, the status of proposed changes to the configuration and the implementation status of approved changes.

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

Control program - A computer program designed to schedule and to supervise the execution of programs in a computer system.

Coverage analysis - The process of determining the degree to which a proposed software verification process activity satisfies its objective.

Database - A set of data, part or the whole of another set of data, consisting of at least one file that is sufficient for a given purpose or for a given data processing system.

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

Data dictionary - The detailed description of data, parameters, variables, and constants used by the system.

Data type - A class of data, characterized by the members of the class and the operations that can be applied to them. Examples are character types and enumeration types.

Deactivated code - Executable object code (or data) which by design is either (a) not intended to be executed (code) or used (data), for example, a part of a previously developed software component, or (b) is only executed (code) or used (data) in certain configurations of the target computer environment, for example, code that is enabled by a hardware pin selection or software programmed options.

Dead code - Executable object code (or data) which, as a result of a design error cannot be executed (code) or used (data) in a operational configuration of the target computer environment and is not traceable to a system or software requirement. An exception is embedded identifiers.

Decision - A Boolean expression composed of conditions and zero or more Boolean operators. A decision without a Boolean operator is a condition. If a condition appears more than once in a decision, each occurrence is a distinct condition.

Decision Coverage - Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once.

Derived requirements - Additional requirements resulting from the software development processes, which may not be directly traceable to higher level requirements.

Emulator - A device, computer program, or system that accepts the same inputs and produces the same output as a given system using the same object code.

Equivalence class - The partition of the input domain of a program such that a test of a representative value of the class is equivalent to a test of other values of the class.

Error - With respect to software, a mistake in requirements, design or code.

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

Failure condition - The effect on the aircraft and its occupants both direct and consequential, caused or contributed to by one or more failures, considering relevant adverse operational and environmental conditions. A failure condition is classified according to the severity of its effect as defined in FAA AC 25.1309-1A or JAA AMJ 25.1309.

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

Fault tolerance - The built-in capability of a system to provide continued correct execution in the presence of a limited number of hardware or software faults.

Formal methods - Descriptive notations and analytical methods used to construct, develop and reason about mathematical models of system behavior.

Hardware/software integration - The process of combining the software into the target computer.

High-level requirements - Software requirements developed from analysis of system requirements, safety-related requirements, and system architecture.

Host computer - The computer on which the software is developed.

Independence - Separation of responsibilities which ensures the accomplishment of objective evaluation. (1) For software verification process activities, independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified, and a tool(s) may be used to achieve an equivalence to the human verification activity. (2) For the software quality assurance process, independence also includes the authority to ensure corrective action.

Integral process - A process which assists the software development processes and other integral processes and, therefore, remains active throughout the software life cycle. The integral processes are the software verification process, the software quality assurance process, the software configuration management process, and the certification liaison process.

Interrupt - A suspension of a task, such as the execution of a computer program, caused by an event external to that task, and performed in such a way that the task can be resumed.

Low-level requirements - Software requirements derived from high-level requirements, derived requirements, and design constraints from which source code can be directly implemented without further information.

Means of compliance - The intended method(s) to be used by the applicant to satisfy the requirements stated in the certification basis for an aircraft or engine. Examples include statements, drawings, analyses, calculations, testing, simulation, inspection, and environmental qualification. Advisory material issued by the certification authority is used if appropriate.

Media - Devices or material which act as a means of transferal or storage of software, for example, programmable read-only memory, magnetic tapes or discs, and paper.

Memory device - An article of hardware capable of storing machine-readable computer programs and associated data. It may be an integrated circuit chip, a circuit card containing integrated circuit chips, a core memory, a disk, or a magnetic tape.

Modified Condition/Decision Coverage - Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

Monitoring - (1) [Safety] Functionality within a system which is designed to detect anomalous behavior of that system. (2) [Quality Assurance] The act of witnessing or inspecting selected instances of test, inspection, or other activity, or records of those activities, to assure that the activity is under control and that the reported results are representative of the expected results. Monitoring is usually associated with activities done over an extended period of time where 100% witnessing is considered impractical or unnecessary. Monitoring permits authentication that the claimed activity was performed as planned.

Multiple-version dissimilar software - A set of two or more programs developed separately to satisfy the same functional requirements. Errors specific to one of the versions are detected by comparison of the multiple outputs.

Object Code - A low-level representation of the computer program not usually in a form directly usable by the target computer but in a form which includes relocation information in addition to the processor instruction information.

Part number - A set of numbers, letters or other characters used to identify a configuration item.

Patch - A modification to an object program, in which one or more of the planned steps of re-compiling, re-assembling or re-linking is bypassed. This does not include identifiers embedded in the software product, for example, part numbers and checksums.

Process - A collection of activities performed in the software life cycle to produce a definable output or product.

Product service history - A contiguous period of time during which the software is operated within a known environment, and during which successive failures are recorded.

Proof of correctness -A logically sound argument that a program satisfies its requirements.

Release - The act of formally making available and authorizing the use of a retrievable configuration item.

Reverse engineering - The method of extracting software design information from the source code.

Robustness -The extent to which software can continue to operate correctly despite invalid inputs.

Simulator - A device, computer program or system used during software verification, that accepts the same inputs and produces the same output as a given system, using object code which is derived from the original object code.

Software - Computer programs and, possibly, associated documentation and data pertaining to the operation of a computer system.

Software architecture - The structure of the software selected to implement the software requirements.

Software change - A modification in source code, object code, executable object code, or its related documentation from its baseline.

Software integration - The process of combining code components.

Software library - A controlled repository containing a collection of software and related data and documents designed to aid in software development, use or modification. Examples include software development library, master library, production library, program library and software repository.

Software life cycle- (1) An ordered collection of processes determined by an organization to be sufficient and adequate to produce a software product. (2) The period of time that begins with the decision to produce or modify a software product and ends when the product is retired from service.

Software partitioning - The process of separating, usually with the express purpose of isolating one or more attributes of the software, to prevent specific interactions and cross-coupling interference.

Software product - The set of computer programs, and associated documentation and data, designated for delivery to a user. In the context of this document, this term refers to software intended for use in airborne applications and the associated software life cycle data.

Software requirement - A description of what is to be produced by the software given the inputs and constraints. Software requirements include both high-level requirements and low-level requirements.

Software tool - A computer program used to help develop, test, analyze, produce or modify another program or its documentation. Examples are an automated design tool, a compiler, test tools and modification tools.

Source code - Code written in source languages, such as assembly language and/or high level language, in a machine-readable form for input to an assembler or a compiler.

Standard - A rule or basis of comparison used to provide both guidance in and assessment of the performance of a given activity or the content of a specified data item.

Statement coverage - Every statement in the program has been invoked at least once.

Static analyzer - A software tool that helps to reveal certain properties of a program without executing the program.

Structure - A specified arrangement or interrelation of parts to form a whole.

System - A collection of hardware and software components organized to accomplish a specific function or set of functions.

System architecture - The structure of the hardware and the software selected to implement the system requirements.

System safety assessment - An ongoing, systematic, comprehensive evaluation of the proposed system to show that relevant safety-related requirements are satisfied.

System safety assessment process - Those activities which demonstrate compliance with airworthiness requirements and associated guidance material, such as, JAA AMJ/FAA AC 25.1309. The major activities within this process include: functional hazard assessment, preliminary system safety assessment, and system safety assessment. The

rigor of the activities will depend on the criticality, complexity, novelty, and relevant service experience of the system concerned.

Task - The basic unit of work from the standpoint of a control program.

Test case - A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Testing - The process of exercising a system or system component to verify that it satisfies specified requirements and to detect errors.

Test procedure - Detailed instructions for the set-up and execution of a given set of test cases, and instructions for the evaluation of results of executing the test cases.

Tool qualification - The process necessary to obtain certification credit for a software tool within the context of a specific airborne system.

Traceability - The evidence of an association between items, such as between process outputs, between an output and its originating process, or between a requirement and its implementation.

Transition criteria - The minimum conditions, as defined by the software planning process, to be satisfied to enter a process.

Validation -The process of determining that the requirements are the correct requirements and that they are complete. The system life cycle process may use software requirements and derived requirements in system validation.

Verification - The evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards provided to that process.

THIS PAGE INTENTIONALLY LEFT BLANK

A P P E N D I C E S

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A BACKGROUND OF DOCUMENT DO-178

1.0 Prior Document Version History

In May 1980, the Radio Technical Commission for Aeronautics, now RTCA, Inc., established Special Committee 145 (SC-145), "Digital Avionics Software", to develop and document software practices that would support the development of software-based airborne systems and equipment. The European Organisation for Civil Aviation Electronics, now the European Organisation for Civil Aviation Equipment (EUROCAE), had previously established Working Group 12 (WG-12) to produce a similar document and, in October 1980, was ready to publish document ED-35, "Recommendations on Software Practice and Documentation for Airborne Systems." EUROCAE elected to withhold publication of its document and, instead, to work in concert with RTCA to develop a common set of guidelines. SC-145 produced RTCA Document DO-178, "Software Considerations in Airborne Systems and Equipment Certification", which was approved by the RTCA Executive Committee and published by RTCA in January 1982. EUROCAE published ED-12 shortly thereafter.

Early in 1983 the RTCA Executive Committee determined that DO-178 should be revised to reflect the experience gained in the certification of the aircraft and engines containing software-based systems and equipment. It established Special Committee 152 (SC-152) for this purpose**.**

As a result of this committee's work, a revised RTCA document, DO-178A, "Software Considerations in Airborne Systems and Equipment Certification", was published in 1985. Shortly thereafter, EUROCAE published ED-12A, which was identical in technical content to DO-178A.

2.0 RTCA / EUROCAE Committee Activities in the Production of This Document

In early 1989, the Federal Aviation Administration (FAA) formally requested that RTCA establish a Special Committee for the review and revision of DO-178A. Since its release in 1985, the aircraft manufacturers, the avionics industry and the certification authorities throughout the world have used DO-178A or the equivalent EUROCAE ED-12A as the primary source of the guidelines to determine the acceptability of systems and equipment containing software.

However, rapid advances in software technology, which were not envisioned by SC-152, and differing interpretations which were applied to some crucial areas, indicated that the guidelines required revision. Accordingly, an RTCA ad hoc group was formed with representatives from ARINC, the Airline Pilots Association, the National Business Aircraft Association, the Air Transport Association and the FAA to consider the FAA request. The group reviewed the issues and experience associated with the application of DO-178A and concluded that a Special Committee should be authorized to revise DO-178A. The RTCA Executive Committee established Special Committee 167 (SC-167) during the autumn of 1989 to accomplish this task, and agreed to these Terms of Reference:

"Special Committee 167 shall review and revise, as necessary, RTCA Document DO-178A, "Software Considerations in Airborne Systems and Equipment Certification."

GUIDANCE:

SC-167 should recognize the dynamic, evolving environment for software requirements, software design, code generation, testing and documentation; and formulate a revised document that can accommodate this environment while recommending suitably rigorous techniques. SC-167 should also recognize the international implications of this document and, therefore, should establish a close working relationship with EUROCAE, which has become the normal practice in RTCA committees. To accomplish this revision, the Special Committee should consider the experience gained through the field application of the guidance material contained in DO-178 and DO-178A,

as well as the results of recent research in software engineering. The Special Committee should focus this review to address these areas:

1. Examine existing industry and government standards and consider for possible adaptation or reference, where relevant.
2. Assess the adequacy of existing software levels and the associated nature and degree of analysis, verification, test and assurance activities. The revised process criteria should be structured to support objective compliance demonstration.
3. Examine the criteria for tools to be used for certification credit, for example, tools for software development, software configuration management and software verification.
4. Examine the certification criteria for previously developed software, off-the-shelf software, databases and user-modifiable software for the system to be certified.
5. Examine the certification criteria for architectural and methodical strategies used to reduce the software level or to provide verification coverage, for example, partitioning and dissimilar software.
6. Examine configuration control guidelines, software quality assurance guidelines and identification conventions; and their compatibility with existing certification authority requirements for type certification, in-service modifications and equipment approval.
7. Consider the impact of new technology, such as, modular architecture, data loading, packaging, and memory technology.
8. Examine the need, content and delivery requirements of all documents, with special emphasis on the Software Accomplishment Summary.
9. Define and consider the interfaces between the software and system life cycles.
10. Review the criteria associated with making pre- and post-certification changes to a system.
11. Consider the impact of evolutionary development and other alternative life cycles to the model implied by DO-178A.

EUROCAE WG-12 was re-established to work with SC-167 and, to accomplish the task, five joint RTCA/EUROCAE working groups were formed to address these topics:

1. Documentation Integration and Production.
2. System Issues.
3. Software Development.
4. Software Verification.
5. Software Configuration Management and Software Quality Assurance.

The cooperative efforts of SC-167 and WG-12 culminated in the publication of RTCA document DO-178B and EUROCAE document ED-12B.

3.0 Summary Of Differences between DO-178B and DO-178A

This is a complete rewrite of DO-178A.

It is suggested that the entire document be read before using any of the sections or tables in isolation.

DO-178B is primarily a process-oriented document. For each process, objectives are defined and a means of satisfying these objectives are described. A description of the software life cycle data which shows that the objectives have been satisfied is provided. The objectives for each process

are summarized in tables and the effect of software level on the applicability and independence of these objectives is specified in the tables. The variation by software level in configuration management rigor for the software life cycle data is also in the tables.

DO-178B recognizes that many different software life cycles are acceptable for developing software for airborne systems and equipment. DO-178B emphasizes that the chosen software life cycle(s) should be defined during the planning for a project. The processes which comprise a software development project, no matter which software life cycle was chosen, are described. These processes fall into three categories: the software planning process, the software development processes, which include software requirements, software design, software coding and integration; and the integral processes which include software verification, software quality assurance, software configuration management and certification liaison. Integral processes are active throughout the software life cycle. DO-178B requires that criteria which control the transitions between software life cycle processes should be established during the software planning process.

The relationship between the system life cycle processes and software life cycle processes is further defined. Failure conditions associated with functions which are to be implemented in software need to be considered during the system safety assessment process. This is the basis for establishing the software level. The software levels are now Level A, Level B, Level C, Level D, and Level E.

The software verification section emphasizes requirements-based testing, which is supplemented with structural coverage.

The items that are to be delivered and other data items are further defined, as well as the configuration management required.

A section covering additional topics has been added to provide guidance in areas previously not addressed in DO-178A. These topics include the use of previously developed software, tool qualification, and the use of alternative methods, including formal methods, exhaustive input testing, multiple-version dissimilar software verification, software reliability models, and product service history.

The glossary of terms has been reviewed for redundant or conflicting terminology and, where possible, industry-accepted definitions adopted.

The guidelines of this document were compared with international standards: ISO 9000-3 (1991), "Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software," and IEC 65A (Secretariat)122 (Draft - November 1991), "Software for Computers in the Application of Industrial Safety-Related Systems." These guidelines are considered to generally satisfy the intent of those standards.

A comprehensive index is provided to assist in the application of this document.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B COMMITTEE MEMBERSHIP

Chairmen:

RTCA SC-167:	Roger A. Seeman	Boeing
EUROCAE WG-12	Daniel J. Hawkes	UK CAA

Secretaries:

RTCA SC-167:	Michael DeWalt	FAA
EUROCAE WG-12	Jean-Michel Nogué	Aerospatiale

Working/Sub-Group Chairmen/Secretaries:

Group 1		Group 4	
B. Pflug	Boeing	J. Angermayer	Bendix/King ATAD
		J-M. Astruc	Veridatas
Group 2		G. Finelli	NASA
D. Allen	Boeing	J. Williams	FAA
M. DeWalt	FAA		
J-M. Nogué	Aerospatiale		
		Group 5	
Group 3		T. Drake	British Aerospace
J. Krodel	Pratt & Whitney	R. Hannan	Smiths Industries
N. Smith	British Aerospace	M. Kress	Boeing

Editorial Group:

A. Coupier	CEAT	B. Pflug	Boeing
R. Hannan	Smiths Industries	C. Secher	DGAC France
E. Kosowski	Rockwell Collins	N. Smith	British Aerospace
T. Kraft	FAA	J. Stesney	Litton Aero
F. Moyer	Rockwell Collins	W. Struck	Boeing
J-M. Nogué	Aerospatiale	L. Tripp	Boeing
K. Peterson	Honeywell	B. Weiser	Honeywell

Federal Aviation Administration Representative:

G. McIntyre — FAA Research and Development

RTCA Representatives	EUROCAE Representative
J. Lohr	B. Perret
D. Watrous	

MEMBERS

Allen, D.	Boeing
Ammann, P.	George Mason University
Anderson, C.	Sundstrand
Angermayer, J.	Bendix/King
Anton, T.	ARINC
Ashpole, R.	British Aerospace
Ashworth, D.	Teledyne Avionics
Astruc, J.	Veridatas
Awad, S.	Canadair
Balivet, E.	SNECMA
Barton, D.	Intermetrics
Beaman, B.	Woodward Governor
Bearn, W.	SDC ASD
Beaulieu, G.	GE Aircraft Engines
Beijard, J.	CEAT
Belcher, G.	GEC Avionics, Ltd.
Bennet, P.	CSE
Berna, P.	Sextant Avionique
Bisiaux, M.	Eurocontrol
Blackburn, M.	Bendix/King ATAD
Blair, K.	Bendix/King
Bosco, C.	FAA
Bowers, R.	Air Transport Association
Bradley, S.	Mitre
Branch, C.	Bendix/King ATAD
Brazell, R.	Vevas Inc.
Bretschneider, M.	Deutsche Airbus
Brewer, A.	Softech
Brown, P.	UK MOD
Brown, W.	Sundstrand
Burtenshaw, G.	UK CAA
Byers, P.	Smith Associates
Caling, J.	Avtech
Campbell, J.	US Coast Guard
Carré, B.	Program Validation, Ltd
Castonguay, R.	ELDEC
Chandler, J.	Lucas Aerospace Ltd.
Cheney, N.	Hamilton Standard
Clemence, G.	Chandler Evans
Climie, B.	Honeywell (Consult.)
Coffman, K.	Simmonds
Cohen, N.	Honeywell
Coleman, J.	Hamilton Standard
Coley, D.	Northstar Avionics
Collins, B.	Dowty Controls
Coupier, A.	CEAT
Crout, J.	Gables Engineering, Inc.
Cucchiara, P.	Kollsman
Davidson, D.	Honeywell
Decker, B.	Douglas Aircraft
Dehlin, G.	Honeywell
Deller, S.	Verdix
Delong, C.	Westar
DeWalt, M.	FAA
Dewshi, M.	Dowty Controls
Dick, A.	GEC Ferranti
Dolman, W.	Lucas Aerospace
Drake, T.	British Aerospace
Drtil, H.	Bodenseewerk Geraetetechnik
Dunham, J.	Research Triangle
Dunst, A.	Smiths Industries
Durgeat, E.	Sextant Avionique
Dvorak, E.	FAA
Eckmann, B.	Microcomputer
Ehrhart, M.	FAA
Fakhouri, O.	Rogerson Kratos
Farrell, J.	Westinghouse
Ferrell, T.	Boeing
Fieldstad, K.	II Morrow Inc
Finelli, G.	NASA
Follet, H.	Aerospatiale
Fortier, S.	Columbia Services Group
Franck, P.	Rockwell Collins
Furstnau, R.	Allison Gas Turbine
Gale, R.	Martin Marietta ATS
Ganz, H.	Airbus Industrie
Garnie, G.	Aike
Garroway, L.	Westinghouse
Gasiorowski, M	Honeywell
Germanicus, P.	Intertechnique
Glasser, J.	Teledyne
Glenn, E.	Teledyne
Goel, A.	Syracuse University
Griffith, E.	II Morrow Inc
Grimshaw, A.	British Aerospace
Hall, B.	Airline Pilots Association
Hall, B.	British Aerospace
Halverson, K.	Rockwell Collins

Hammar, J.	Search Technology
Hannah, J.	Info Spectrum Inc
Hannan, R.	Smiths Industries
Hannert, L.	Honeywell
Harrison, W.	Advanced System Tech.
Hawkes, D.	UK CAA
Heller, P.	Deutsche Airbus
Hendrickson, T.	Boeing
Hill, K.	Boeing
Hopf, R.	GE Aircraft Engines
Hopkins, D.	Pratt & Whitney, Canada
Hostert, S.	Rockwell Collins
Hung, B.	MITRE
Hutchinson, S.	ARINC
Iapicca, C.	ALENIA (Aeritalia)
Ion, J.	Lohr Systems
Jack, C.	Rolls-Royce
Janelle, J.	Honeywell
Johnson, R.	Delco Systems Ops
Karis, C.	GE-ACSD
Keizer, K.	Microcomputer
Keller, F.	FAA
Kerr, R.	Honeywell
Kilbane, T.	Westinghouse
Kirby, N.	Boeing
Kleine-Beek, W.	LBA
Klinka, B.	FAA
Kosowski, E.	Rockwell Collins
Kraft, T.	FAA
Kress, M.	Boeing
Krodel, J.	Pratt & Whitney
Kuhl, F.	Mitre
Kuhn, R.	NIST
Kurowsky, R.	US Army
Ladier, G.	Aerospatiale
Lampton, M.	Rockwell Collins
Langumier, P.	DGAC France
LaPietra, P.	GE Aircraft Engines
Laurent, O.	Aerospatiale
Leveson, N.	University of California
Lewis, D.	Boeing
Lillis, M.	Hamilton Standard
Lin, M.	Litton Aero Products
Littlewood, B.	Centre for SW Reliability
Lleres, J.	SNECMA
Lock, H.	ARINC - AEEC
Lohr, J.	RTCA
Luck, H.	Aero-Radui
Marchand, M.	Aerospatiale
Marzke, L.	Kollsman
Mastantvoild, P.	DGAC France
Mattissek, A.	LITEF
Mayor, M.	Boeing of Canada
McCallister, R.	FAA
McCormick, T.	Allied-Signal Canada
McDonald, W.	Sundstrand
McIntyre	FAA R&D
McKinlay, A.	Douglas Aircraft
McKown, B.	Boeing
Meyer, K.	Integsystems
Miller, D.	George Mason University
Mischkot, E.	Garrett Canada
Monavon, M.	SNECMA
Montgomery, S.	Simmonds
Moore, L.	Honeywell
Morrice, G.	GEC Ferranti Defence
Morse, T.	Boeing
Moyer, F.	Rockwell Collins
Mueller, H.	FAA
Myers, J.	Simmonds
Neilan, P.	UK CAA
Nelson, K	Woodward Governor
Newton, I.	GEC Avionics, Ltd.
Nogué, J-M.	Aerospatiale
Norris, J.	ELDEC
Nutaro, J.	Honeywell
Oberg, M.	Bendix/King
Ochs, D.	Cessna Aircraft
Olivier, E.	Hamilton Standard
Oss, J.	Honeywell
Paasch, S.	FAA
Patterson, W.	Westinghouse
Peak, A.	Ametek Aerospace
Pearce, M.	GEC Ferranti Defence
Penny, J.	British Aerospace
Perini, M.	FAA
Perret, B.	EUROCAE
Peterson, K.	Honeywell
Pflug, B.	Boeing
Pharr, J.	Teledyne Avionics
Piper, E.	Douglas Aircraft
Porter, C.	Kollsman
Prisaznuk, P.	ARINC - AEEC
Quinby, G.	Narco Avionics

Ramji, S.	Canadian Marconi
Rao, R.	Transport Canada
Rawas, M.	Beech Aircraft
Reich, A.	Racal Avionics
Reige, J.	Woodward Governor
Reitz, G.	Litton Aero Products
Reynold, B.	Douglas Aircraft
Roberts, F.	Douglas Aircraft
Robinson, P.	Rolls-Royce
Rodgers, D.	Boeing
Roth, T.	Bendix/King
Rowe, W.	Westinghouse
Ruana, R.	Jeppesen
Russell, W.	Air Transport Association
Ryburn, R.	FAA
Saraceni, P.	FAA
Sarich, C.	FAA
Satyen, U.	MITRE
Sayegh, S.	Canadian Marconi
Schad, L.	Boeing
Schirle, P.	Dassault Aviation
Schmitz, S.	Alcatel TITN
Scott, R.	British Aerospace
Secher, C.	DGAC France
Seeman, R.	Boeing
Semmakie, E.	British Airways
Shagnea, A.	Research Triangle
Shahijanian, P.	Litton Aero Products
Shimmin, A.	British Aerospace
Shore, D.	Honeywell
Silver, S.	Litton Aero Products
Simmons, J.	Woodward Governor
Simonetti, A.	Veridatas
Sitz, J.	NASA
Skaves, P.	FAA
Smart, J.	British Aerospace
Smith, N.	British Aerospace
Smith, W.	Rockwell Collins
Spoor, H.	GEC Ferranti
Stack, N.	Litton Aero Products
Stallwitz, C.	Beech Aircraft
Stesney, J.	Litton Aero Products
Stihl, D.	Douglas Aircraft
Stolzenthaler, J.	Hamilton Standard
Struck, W.	Boeing
Summers, P.	Rolls Royce
Teichert, P.	LBA Germany
Temprement, M.	Sextant Avionique
Tessier, B.	FAA
Theodore, A.	Bendix/King ATAD
Thompson, L.	Honeywell
Toldo, D.	SNECMA
Tomasi, J.	Aerospatiale
Tourret, O.	STTE
Tripp, L.	Boeing
Tucker, R.	Teledyne Controls
Tucker, S.	GTE Airfone
Turton, J.	UK MOD
Van Baal, J.	RLD The Netherlands
Van De Hulst, H.	Fokker
Van Houtte, E.	ARINC
Vaughn, R.	FAA
Vincent, J.	UK CAA
Wade, M.	FAA
Walker, T.	GE Aircraft Engines
Ward, J.	Gulfstream Aero
Watrous, D.	RTCA
Watson, J.	Bendix/King
Watts, R.	Honeywell
Weadon, T.	Honeywell
Weiser, B.	Honeywell
Wells, R.	Boeing
Wilkinson, M.	Smith Associates
Williams, J.	FAA
Wilson, W.	Litton Aero Products
Wojciech, J.	FAA
Wolf, C.	FAA
Wolf, R.	Teledyne
Wolfley, K.	FAA
Wong, A.	FAA
Wood, B.	CMU/SEI
Woodfield, D.	Litton Aero Products
Wright, K.	Smiths Industries
Wright, C.	FAA
Young K.	Honeywell
Yuen, J.	Canadian Marconi

INDEX OF TERMS

APPENDIX C

This index includes these terms:

- Items identified in the glossary.
- Terms of Reference.
- Useful alternative nomenclature for important concepts

All indexed terms are followed by page number references. The underlined page number(s) indicates the page where the term is defined.

---

verification - see software verification

APPENDIX D IMPROVEMENT SUGGESTION FORM

Name: _____ Company Name: _____

Address: _____

City: _____ State, Postal Code, Country: _____

Phone: _____ Date: _____

Document : **DO-178/ED-12 Revision B** Sec: _____ Page: _____ Line: _____

[ ] Documentation error (Format, punctuation, spelling)

[ ] Content error

[ ] Enhancement or refinement

Rationale (Describe the error or justification for enhancement): _____

Proposed change (Attach marked-up text or proposed rewrite): _____

Please provide any general comments for improvement of this document:

Return completed form to:

RTCA, Inc.
Attention: DO-178B
1140 Connecticut Ave. NW Suite 1020
Washington, D. C. 20036 USA

OR

EUROCAE
Attention: Secretary-General
17 Rue Hamelin
Paris Cedex 75783 France

THIS PAGE INTENTIONALLY LEFT BLANK