DO-178B

201372235 김영승
201372237 이선아

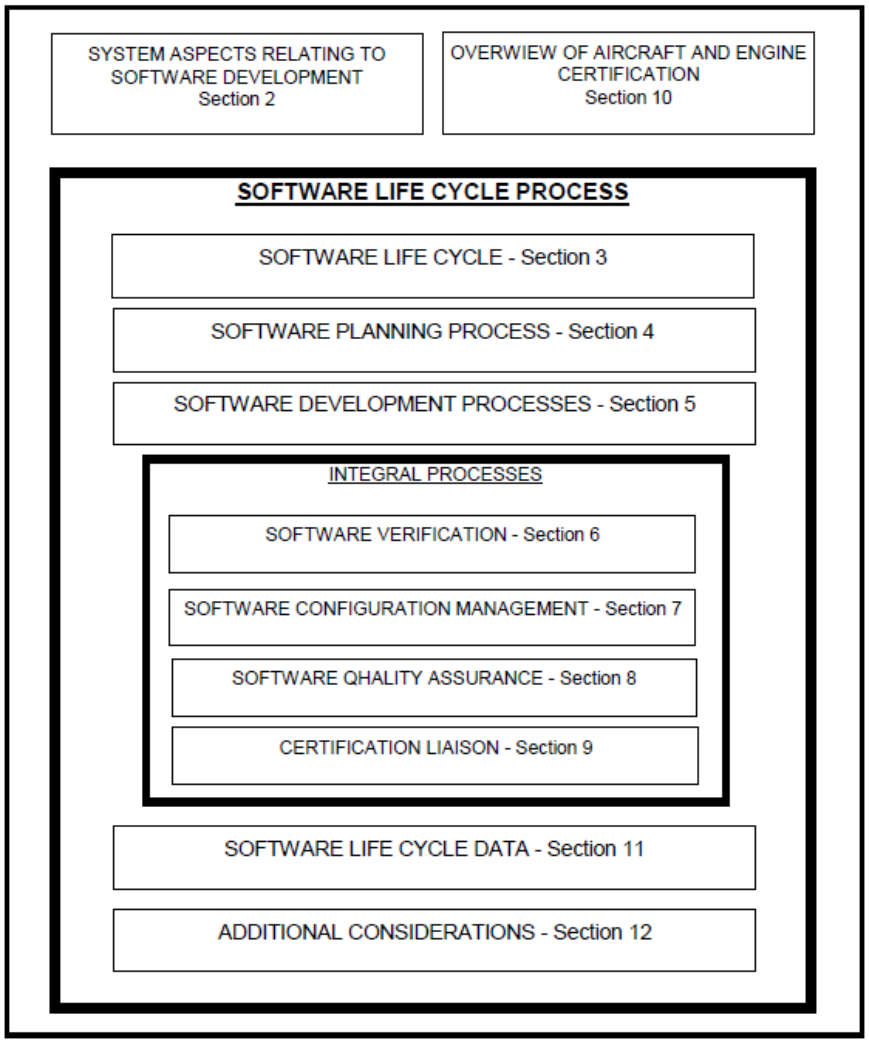
Contents

- Introduction
- Standard
  - SECTION 1 INTRODUCTION
  - SECTION 2 SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT
  - SECTION 3 SOFTWARE LIFE CYCLE
  - SECTION 4 SOFTWARE PLANNING PROCESS
  - SECTION 5 SOFTWARE DEVELOPMENT PROCESSES
  - SECTION 6 SOFTWARE VERIFICATION PROCESS
  - SECTION 7 SOFTWARE CONFIGURATION MANAGEMENT PROCESS
  - SECTION 8 SOFTWARE QUALITY ASSURANCE PROCESS
  - SECTION 9 CERTIFICATION LIAISON PROCESS
  - SECTION 10 OVERVIEW OF AIRCRAFT AND ENGINE CERTIFICATION
  - SECTION 11 SOFTWARE LIFE CYCLE DATA
  - SECTION 12 ADDITIONAL CONSIDERATIONS

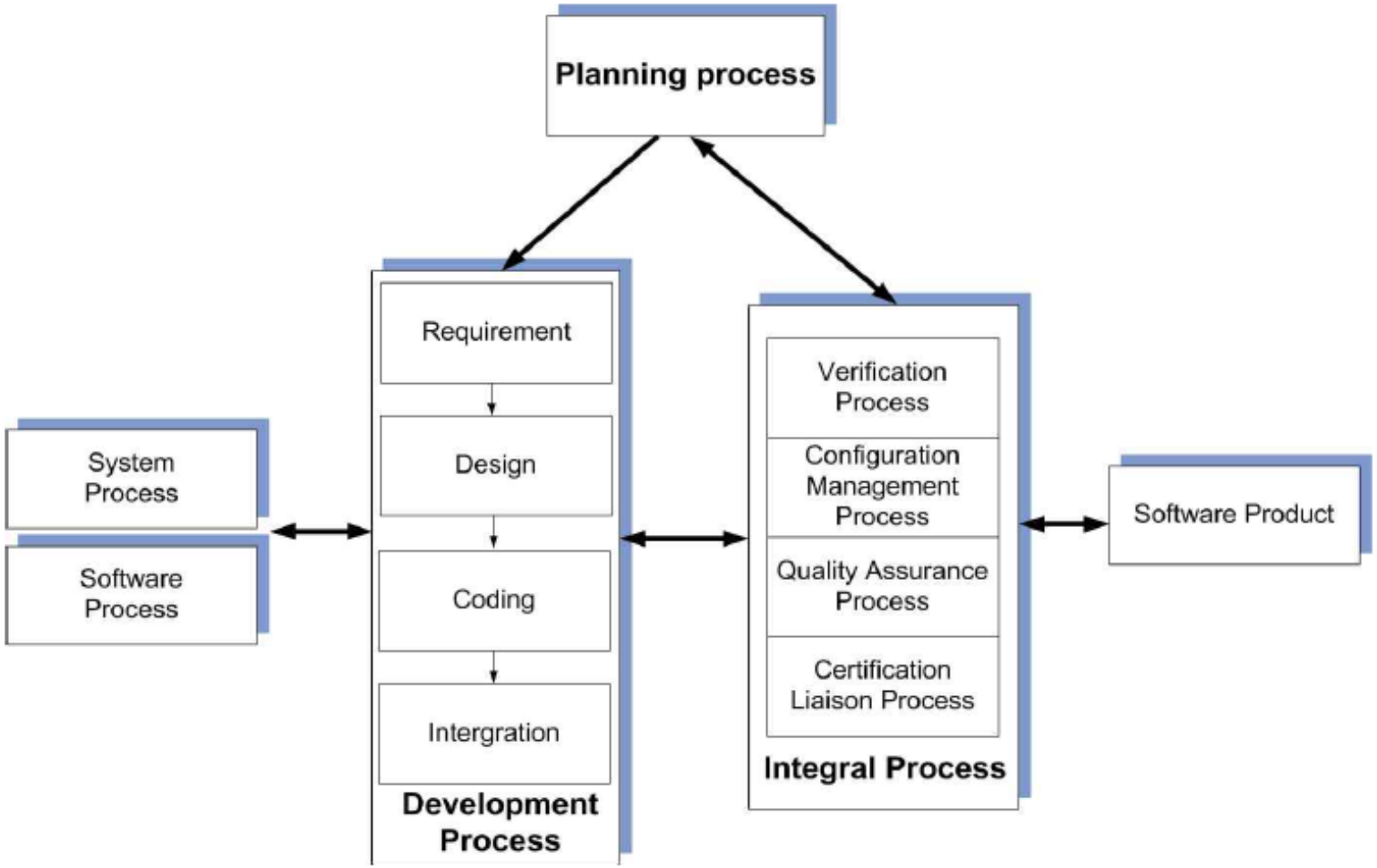
Introduction

- **DO-178B?**
  - Software Considerations in Airborne System and Equipment Certification
  - 항공기 소프트웨어 오작동으로 인한 항공기 사고 위험을 최소화하기 위해 항공기 소프트웨어 안정성에 대한 국제적인 인증 표준

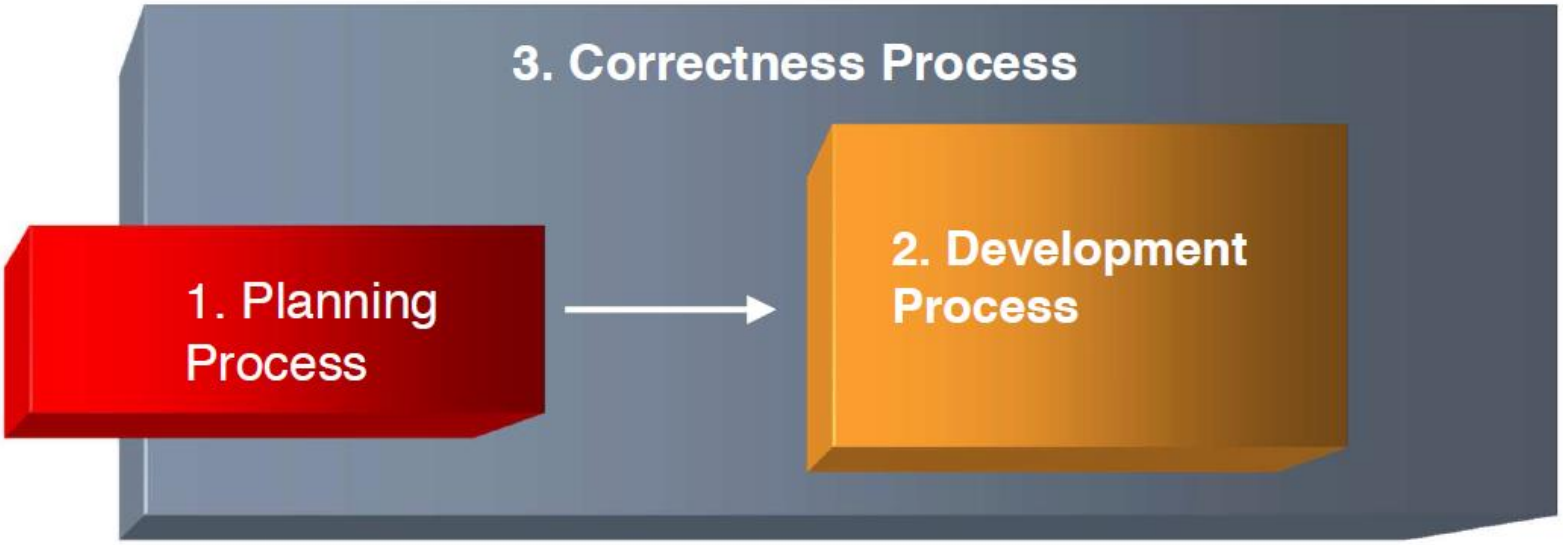
Overview


SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT Section 2
OVERWIEW OF AIRCRAFT AND ENGINE CERTIFICATION Section 10
SOFTWARE LIFE CYCLE PROCESS
SOFTWARE LIFE CYCLE - Section 3
SOFTWARE PLANNING PROCESS - Section 4
SOFTWARE DEVELOPMENT PROCESSES - Section 5
INTEGRAL PROCESSES
SOFTWARE VERIFICATION - Section 6
SOFTWARE CONFIGURATION MANAGEMENT - Section 7
SOFTWARE QHALITY ASSURANCE - Section 8
CERTIFICATION LIAISON - Section 9
SOFTWARE LIFE CYCLE DATA - Section 11
ADDITIONAL CONSIDERATIONS - Section 12

DO-178B


Planning process
Requirement
Design
Coding
Intergration
Development Process
System Process
Software Process
Verification Process
Configuration Management Process
Quality Assurance Process
Certification Liaison Process
Integral Process
Software Product

3 Key-Process



5 Key Plans

1. PSAC	2. SQAP	3. SCMP	4. SWDP	5. SWVP

- PSAC: Plan for Software Aspects of Certification
- SQAP: Software Quality Assurance Plan
- SCMP: Software Configuration Management Plan
- SWDP: Software Development Plan
- SWVP: Software Verification Plan

*** Plus 3 Standards: Requirements, Design and Coding

SECTION 2

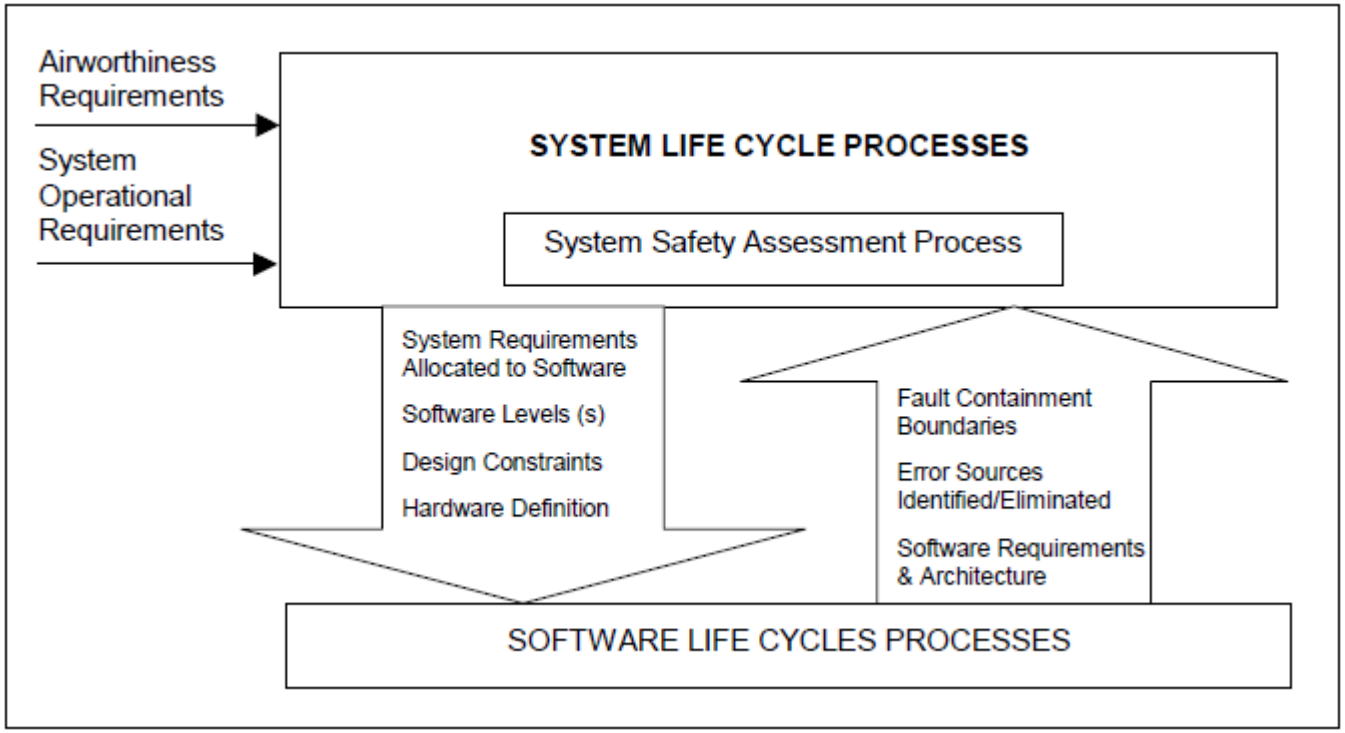
SYSTEM ASPECTS RELATING TO SOFTWARE DEVELOPMENT

SECTION 2

- Software Life Cycle Process를 이해하기 위해 필요한 System Life Cycle Process의 ㅇㅇ을 살펴본다.
- 2.1 Information flow between system and software life cycle processes
- 2.2 Failure condition and software level
- 2.3 System architectural considerations
- 2.4 System considerations for user-modifiable software, option-selectable software and commercial off-the-shelf software
- 2.5 System design considerations for field-loadable software
- 2.6 System requirements considerations for software verification
- 2.7 Software considerations in system verification

2.1 (Information flow between system and software life cycle processes)

- an overview of the safety aspects of the information flow between system life cycle processes and the software life cycle processes.



2.1 (Information flow between system and software life cycle processes)

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

2.1 (Information flow between system and software life cycle processes)

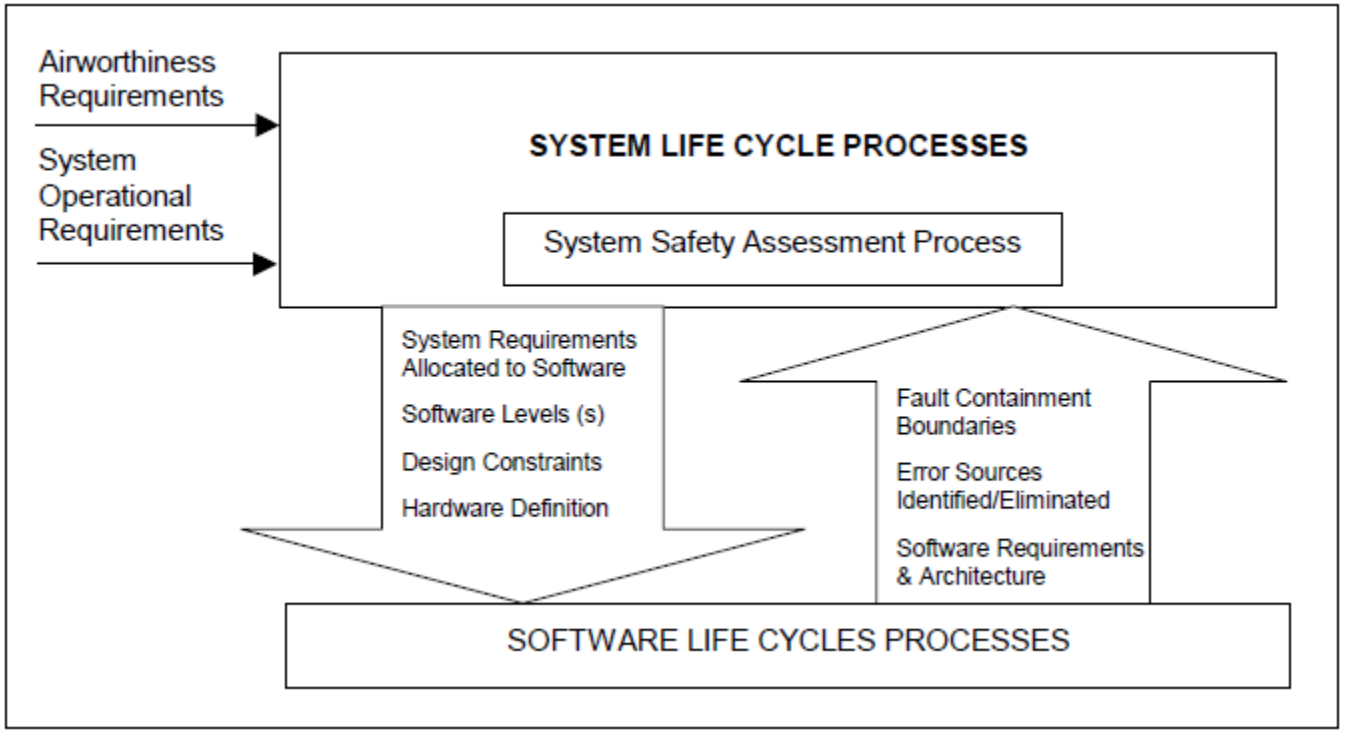
- 2.1.1 Information flow from System Processes to Software Processes
  - System Safety Assessment Process
    - System의 failure conditions을 결정하고 분류함.
    - Safety-related requirements 정의함.
    - Safety-related requirements를 만족하는지 확인하기 위해 system design의 결과를 분석함.
  - Safety-related requirement
    - The system description and hardware definition
    - Certification requirements
    - System requirements allocated to software
    - Software level, failure conditions, related functions
    - Safety strategies, design constraints

2.1 (Information flow between system and software life cycle processes)

- 2.1.2 Information Flow from Software Processes to System Processes
  - System Safety Assessment Process
    - system safety에 대한 software design과 implementation의 영향을 결정함.
    - system requirement와 software design data사이의 traceability
- 미완성…

SECTION 2

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



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 (Failure condition and software level)

- 2.2.1 Failure Condition Categorization
  - 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 (Failure condition and software level)

- 2.2.1 Failure Condition Categorization
- 2.2.2 Software Level Definitions

결함조건 (ARP 4761)	정의	레벨
치명 결함 (catastrophic)	계속적인 안전 비행 및 착륙을 못하는 결함 상태	A
위험/심각-중 결함 (hazardous/ severe-major)	아래와 같은 분야의 잘못된 운용 상태에 대처할 수 있는 항공기의 능력 또는 승무원의 능력을 감소시키는 결함 상태 (1) 안전 여유 또는 기능적 능력의 상당한 감소 (2) 비행 승무원이 그들의 임무를 정확하게 또는 완전히 수행하는데 집중할 수 없을 정도의 육체적 피로 또는 과도한 업무 부하 (3) 소수의 탑승객에게 심각하거나 치명적인 손상을 포함하여 탑승객에 미치는 악영향	B
중 결함(major)	다음과 같은 분야의 잘못된 운용 상태에 대처할 수 있는 항공기의 능력 또는 승무원의 능력을 감소시키는 결함 상태. 예를 들면, 안전 여유 또는 기능적 능력의 상당한 감소, 승무원 업무부하의 심각한 증가 또는 승무원의 효율을 떨어뜨릴 수 있는 상태 또는 손상을 포함하여 승객을 불편하게 하는 것	C
경 결함(minor)	항공기의 안전을 심각하게 감소시키지 않는 결함 상태이며 승무원들의 능력 범위 내에서 잘 처리할 수 있는 행위를 포함한다. 경 결함 상태는 예를 들면 안전 여유 또는 기능적 능력의 경미한 감소, 통상적 비행 계획의 변경, 또는 탑승객에 약간의 불편함을 끼치는 것 과 같은 비행 승무원 부하의 경미한 증가를 포함 할 수 있다.	D
무 영향 결함 (no effect)	항공기의 운용 능력에 영향을 미치지 않거나 승무원의 업무부하에 영향을 미치지 않는 결함 상태	E

2.3 (System architectural considerations)

- 2.2.3 Software Level Determination
- 2.3 SYSTEM ARCHITECTURAL CONSIDERATIONS
  - 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.
- 2.3.2 Multiple-Version Dissimilar Software
- 2.3.3 Safety Monitoring

2.4 (System considerations for user-modifiable software, optionselectable software and commercial off-the-shelf software)

- 2.4 SYSTEM CONSIDERATIONS FOR USER-MODIFIABLE SOFTWARE, OPTIONSELECTABLE SOFTWARE AND COMMERCIAL OFF-THE-SHELF SOFTWARE

2.5 (System design considerations for field-loadable software)

- 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.

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.
  - 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)

Section 3

SOFTWARE LIFE CYCLE

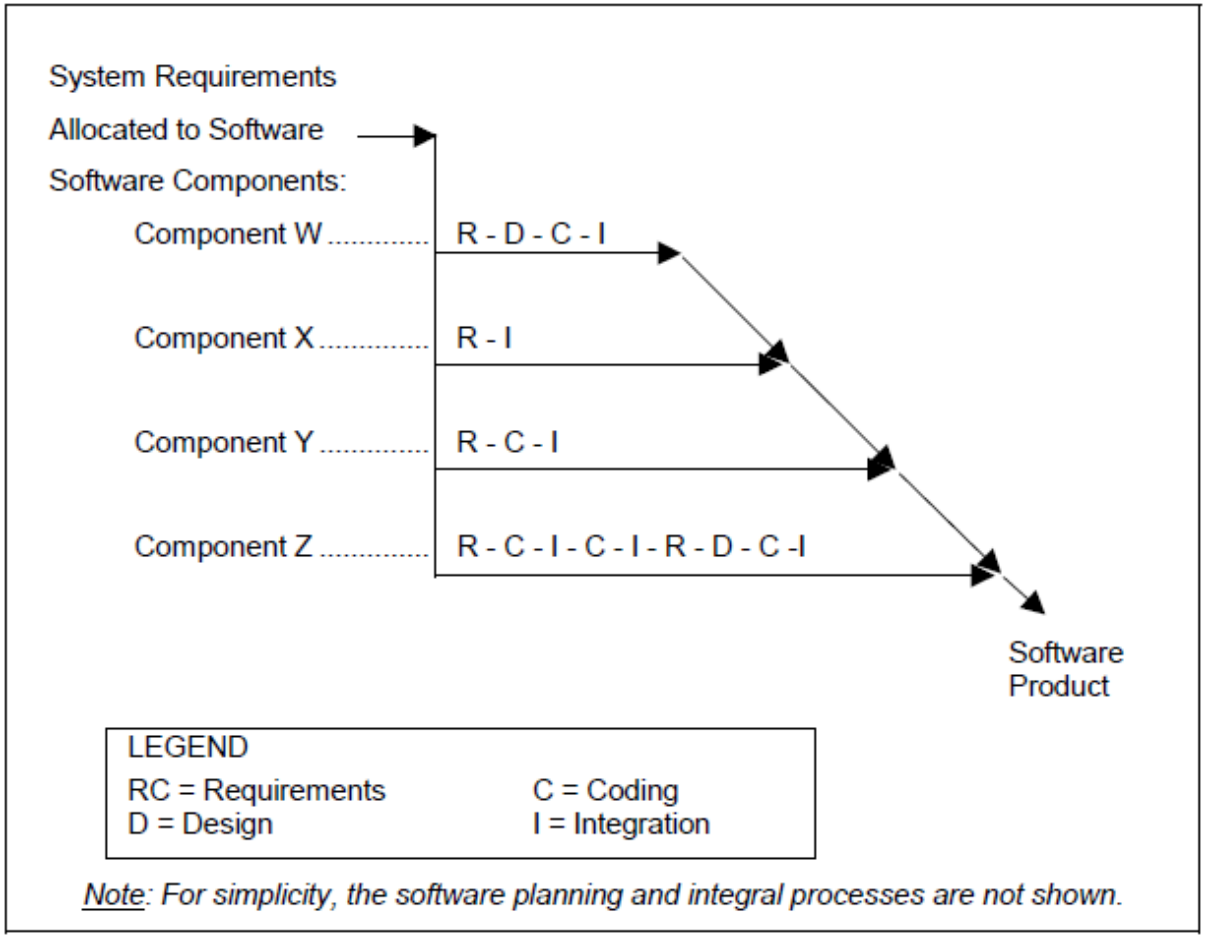
SECTION 3

- 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.

SECTION 3

- 3.1 SOFTWARE LIFE CYCLE PROCESSES
- 3.2 SOFTWARE LIFE CYCLE DEFINITION
- 3.3 TRANSITION CRITERIA BETWEEN PROCESSES

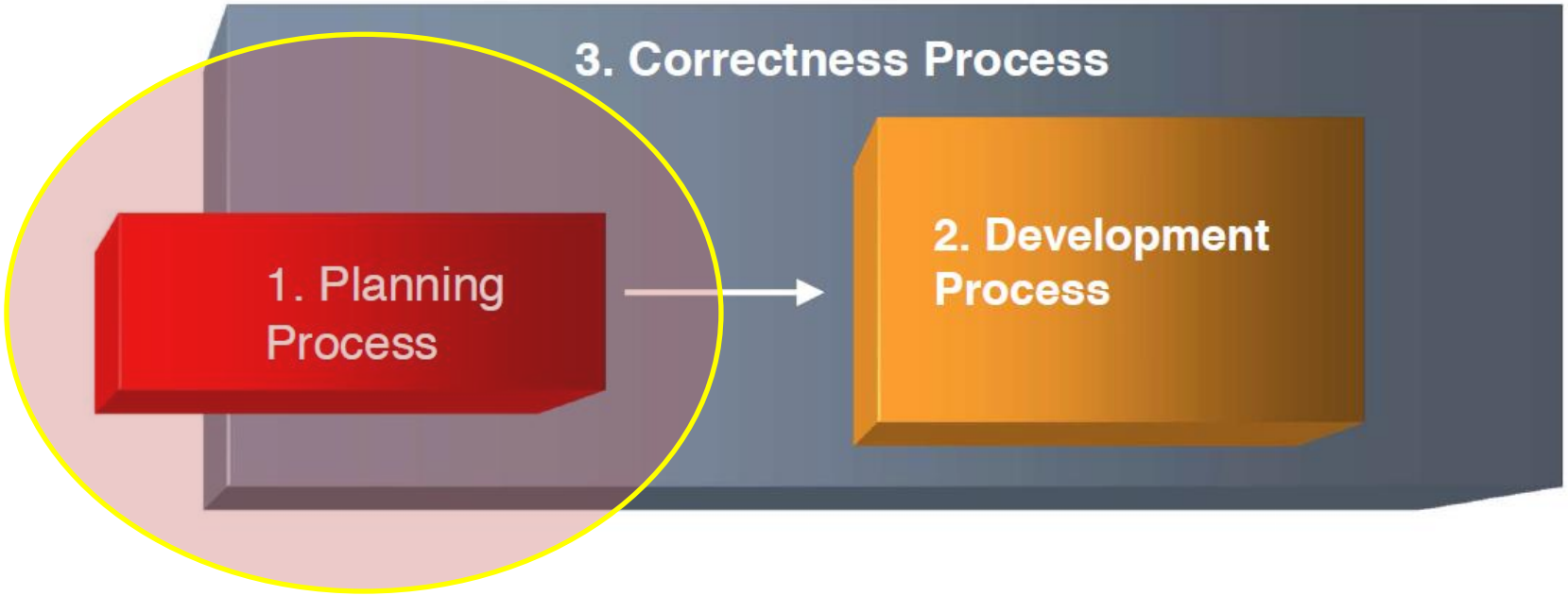
SECTION 3


System Requirements
Allocated to Software
Software Components:
Component W ..... R - D - C - I
Component X ..... R - I
Component Y ..... R - C - I
Component Z ..... R - C - I - C - I - R - D - C -I
Software Product
LEGEND
RC = Requirements
D = Design
C = Coding
I = Integration
Note: For simplicity, the software planning and integral processes are not shown.

Section 4

SOFTWARE PLANNING PROCESS

3 Key-Process



SECTION 4

- 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.

4.1 (Software planning process objectives)

- 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).

4.3 (Software plans)

- 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.

4.3 (Software plans)

- 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.

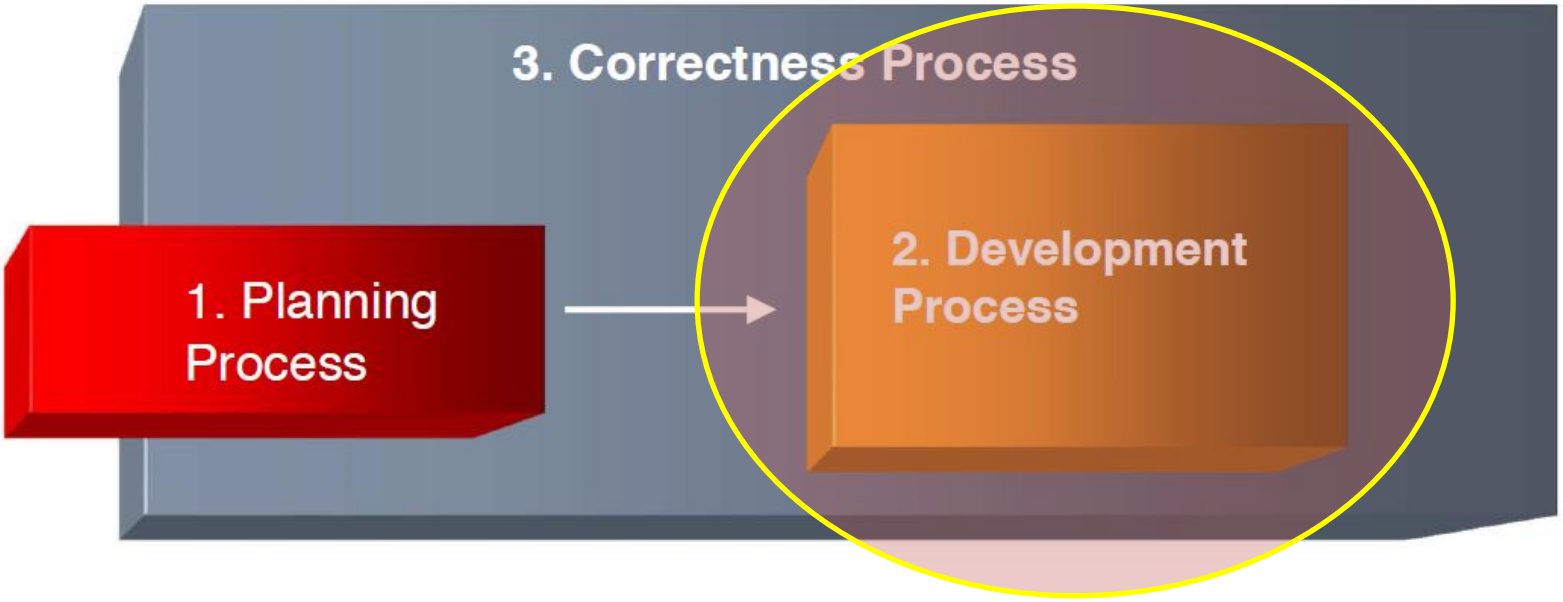
SECTION 4

- 4.4 SOFTWARE LIFE CYCLE ENVIRONMENT PLANNING
  - 4.4.1 Software Development Environment
  - 4.4.2 Language and Compiler Considerations
  - 4.4.3 Software Test Environment
- 4.5 SOFTWARE DEVELOPMENT STANDARDS
- 4.6 REVIEW AND ASSURANCE OF THE SOFTWARE PLANNING PROCESS

Section 5

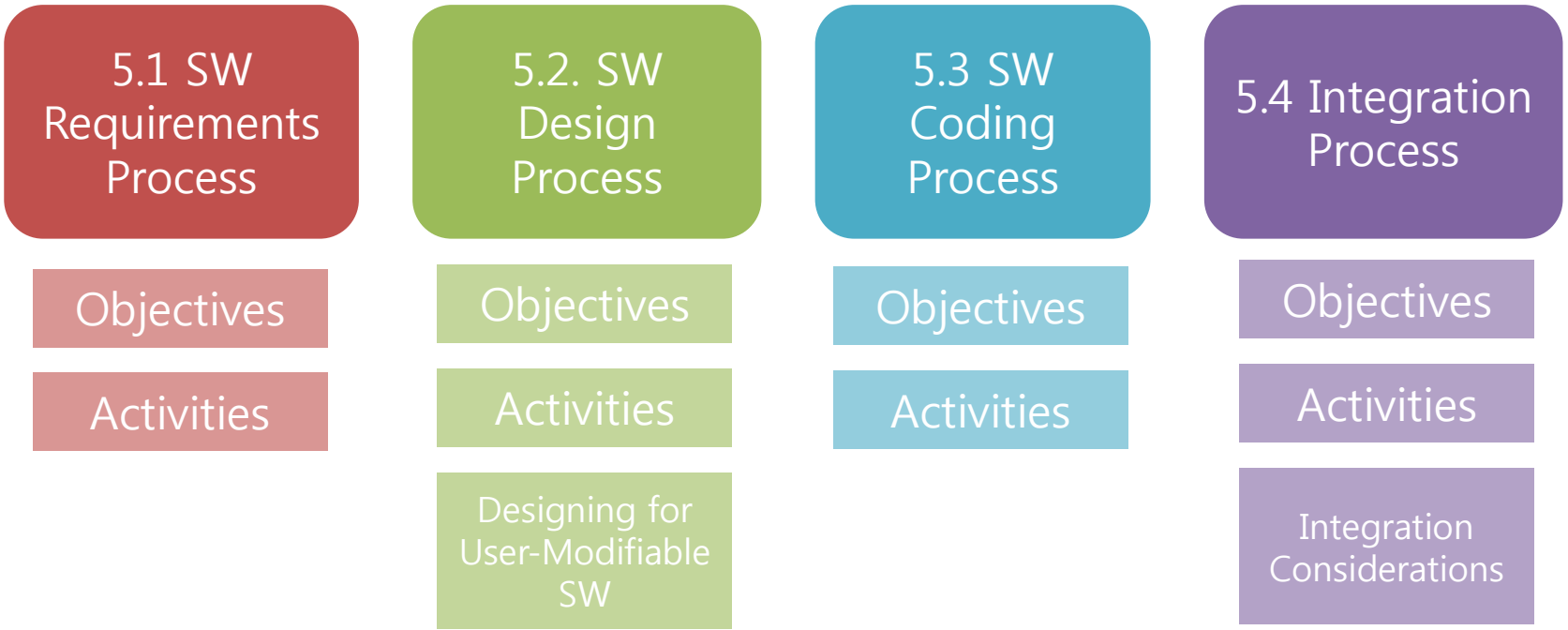
SOFTWARE DEVELOPMENT PROCESSES

3 Key-Process



Section 5

- Software planning process (Section 4) & Software Development Plan (Section 11)에 의해 정의되고, 수행함.
- **Software Development Process**



5.1 (Software Requirements Process)

- 5.1.1 Software Requirements Process **Objectives**
  - 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**
  - 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.

5.1 (Software Requirements Process)

- 5.1.2 Software Requirements Process **Activities** (이 프로세스가 완료되기 위해서)
  - 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)

- 5.2.1 Software Design Process **Objectives**
  - 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**
  - 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. **(5.1.e 와 유사)**
  - b. Derived requirements should be defined and analyzed to ensure that the higher level requirements are not compromised.

5.2 (Software Design Process)

- 5.2.2 Software Design Process Activities
  - 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. **(5.1.B와 유사)**

5.2 (Software Design Process)

- **5.2.3 Designing for User-Modifiable Software**
  - designed and certified to allow for limited modifications by an airline or other user without recertification efforts.
  - Software of any Level can include a modifiable component.
    - 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**
  - Source code is developed that is traceable, verifiable, consistent, and correctly implements low-level requirements.
- 5.3.2 Software Coding Process **Activities**
  - 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**
- 5.4.2 Integration Process **Activities**
- 5.4.3 Integration **Considerations**

Objectives

Activities

Integration Considerations

5.4 (Integration Process)

- 5.4.1 Integration Process **Objectives**
  - a. The Executable Object Code is loaded into the target hardware for hardware/software integration.
- 5.4.2 Integration Process **Activities**
  - 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 (Integration Process)

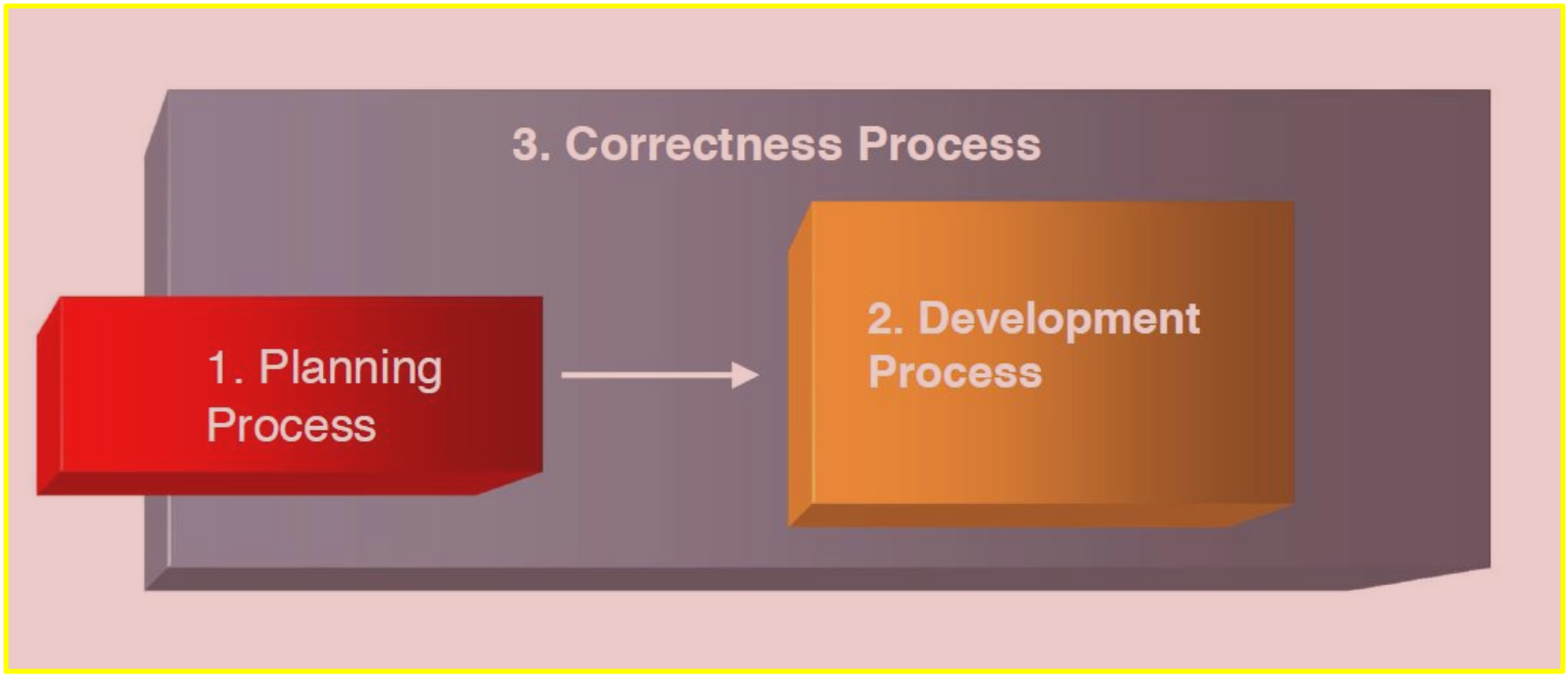
- 5.4.3 Integration Considerations
  - **Deactivated Code and Software Patches.**
  - This can lead to deactivated code that cannot be executed or data that is not used.
  - Patches may be used on a limited, case- by-case basis, for example, to resolve known deficiencies in the software development environment, such as a known compiler problem.
  - When a patch is used, these should be available:
    - Confirmation that the software configuration management process can effectively track the patch.
    - Regression analysis to provide evidence that the patch satisfies all objectives of the software developed by normal methods.
    - Justification in the Software Accomplishment Summary for the use of a patch.

5.5 (Traceability)

- 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.**

Traceability  Verification

3 Key-Process



Section 6

SOFTWARE VERIFICATION PROCESS

SECTION 6

- 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).

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.

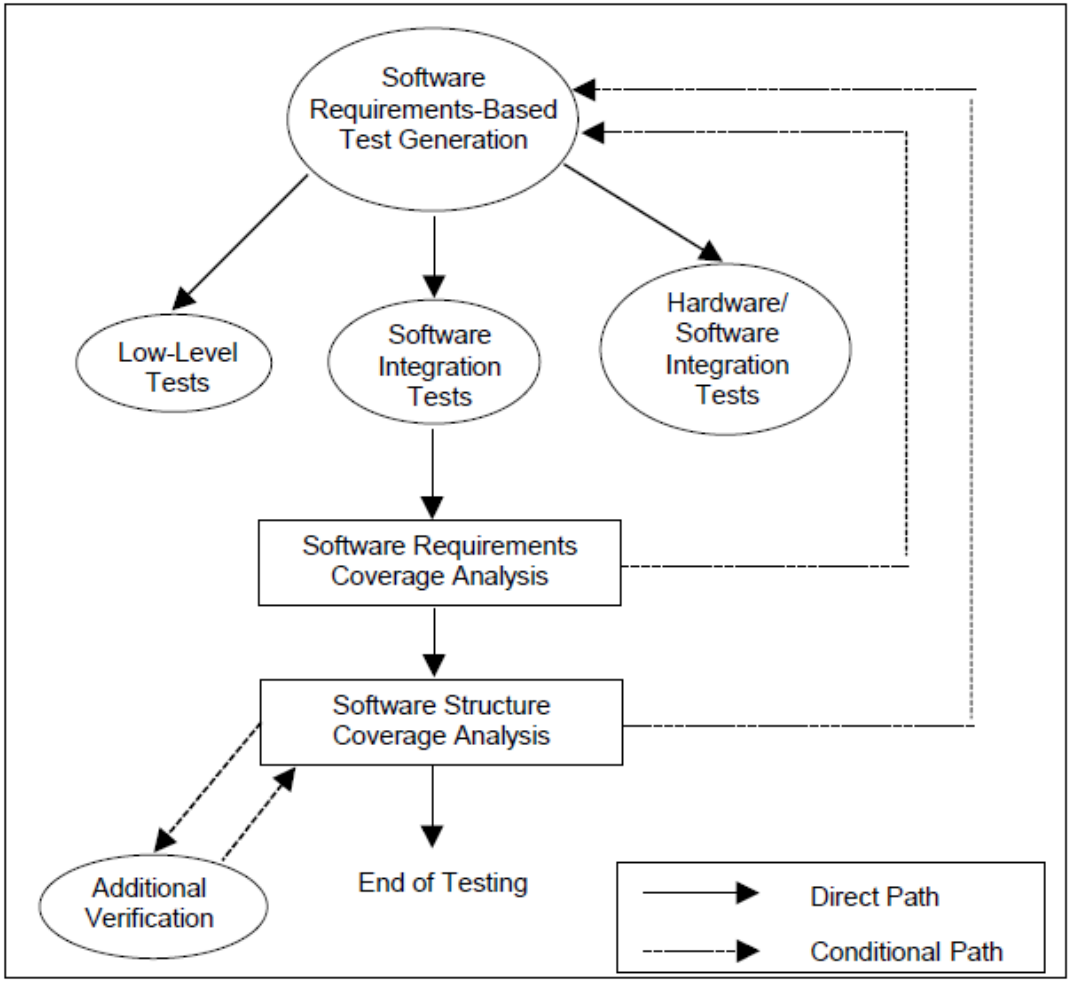
6.2 (Software verification process activities)

- 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.
- 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)

- 6.3.1 Reviews and Analyses of the High-Level Requirements
- 6.3.2 Reviews and Analyses of the Low-Level Requirements
- 6.3.3 Reviews and Analyses of the Software Architecture
- 6.3.4 Reviews and Analyses of the Source Code
- 6.3.5 Reviews and Analyses of the Outputs of the Integration Process
- 6.3.6 Reviews and Analyses of the Test Cases, Procedures and Results

6.4 (Software Testing)



6.4 (Software Testing)

Methods	Objects	Error To Detect
HW / SW 통합 테스팅	High Level Requirements	인터럽트 핸들링, 실행시간 요건, HW 실패에 대한 SW 반응, Data Bus 및 자원 연결 문제, 자체 고장 탐지 실패, 피드백 루프 오류, 메모리관리 HW 비정상 제어, Stack overflow, 필드적재 SW 보증 메커니즘 오류 등
SW 통합 테스팅	SW Requirements / Architecture	변수/상수 비정상 초기화, 매개변수 전달 오류, 데이터 손상(전역데이터), 이벤트 및 동작 순서 오류
Low Level 테스팅	Low Level Requirements	알고리즘 오류, 비정상 루프, 로직 오류, 비정상적인 입력의 조합, 누락/손상 입력에 대한 비정상적 반응, 예외처리 실패, 계산 순서 오류, 알고리즘 정확도/성능 정밀도 오류

6.4 (Software Testing)

- 6.4.2 Requirements-Based Test Case Selection
  - 6.4.2.1 Normal Range Test Cases
  - 6.4.2.2 Robustness Test Cases
- 6.4.3 Requirements-Based Testing Methods
  - a. Requirements-Based Hardware/Software Integration Testing
  - b. Requirements-Based Software Integration Testing
  - c. Requirements-Based Low-Level Testing

6.4 (Software Testing)

- 6.4.4 Test Coverage Analysis
  - 6.4.4.1 Requirements-Based Test Coverage Analysis
  - 6.4.4.2 Structural Coverage Analysis
  - 6.4.4.3 Structural Coverage Analysis Resolution

Section 7

SOFTWARE CONFIGURATION MANAGEMENT PROCESS

SECTION 7

- 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.

7.2 (Software configuration management process activities)

- 7.2.1 Configuration Identification
- 7.2.2 Baselines and Traceability
- 7.2.3 Problem Reporting, Tracking and Corrective Action
- 7.2.4 Change Control
- 7.2.5 Change Review
- 7.2.6 Configuration Status Accounting
- 7.2.7 Archive, Retrieval and Release
- 7.2.8 Software Load Control
- 7.2.9 Software Life Cycle Environment Control

7.3 (Data control categories)

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

7.3 (Data control categories)

SCMProcess Objective	Reference	CC1	CC2
Configuration Identification	7.2.1	o	o
Baselines	7.2.2 a, b, c, d, e	o	
Traceability	7.2.2f, g	o	o
Problem Reporting	7.2.3	o	
Change Control - integrity and identification	7.2.4a, b	o	o
Change Control - tracking	7.2.4c, d, e	o	
Change Review	7.2.5	o	
Configuration Status Accounting	7.2.6	o	
Retrieval	7.2.7a	o	o
Protection against Unauthorized Changes	7.2.7b(1)	o	o
Media Selection, Refreshing, Duplication	7.2.7b(2), (3), (4), c	o	
Release	7.2.7d	o	
Data Retention	7. 2.7e	o	o

Section 8

SOFTWARE CONFIGURATION MANAGEMENT PROCESS

SECTION 8

- Software planning process (Section 4) &
  Software Quality Assurance Plan(Section 11.5)에 의해 정의되고 수행함.
- 8.1 Software Quality Assurance (SQA) Process Object.
- 8.2 Software Quality Assurance (SQA) Process Activity.
- Outputs of the SQA process activities are recorded in Software Quality Assurance Records (subsection 11.19) or other software life cycle data.

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.
  - 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.

8.2 (Software Quality Assurance Process Activities)

- 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.

8.2 (Software Quality Assurance Process Activities)

- 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.
  - (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.

8.2 (Software Quality Assurance Process Activities)

- 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)

- 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, safetyrelated 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.

8.3 (Software Conformity Review)

- 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 reevaluated 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.

Section 9

CERTIFICATION LIAISON PROCESS

SECTION 9

- 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.
- Software Planning Process (section 4) & Plan For Software Aspects Of Certification (Section 11.1)에 의해 정의되고 수행됨.
- 9.1 Means Of Complaince And Planning
- 9.2 Complaince Substantiation
- 9.3 Minimum Software Life Cycle Data That Is Sibmitted To Certification Authority
- 9.4 Software Life Cycle Data Related To Type Design

Section 10

OVERVIEW OF AIRCRAFT AND ENGINE CERTIFICATION

SECTION 10

- 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 Certifiacation Basis
- 10.2 Software Aspects Of Certification
- 10.3 Compliance Determination

Section 11

SOFTWARE LIFE CYCLE DATA

SECTION 11

- 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.

SECTION 11

- 11.1 Plan for Software Aspects of
- 11.2 Software Development
- 11.3 Software Verification
- 11.4 Software Configuration Management
- 11.5 Software Quality Assurance
- 11.6 Software Requirements
- 11.7 Software Design
- 11.8 Software Code
- 11.9 Software Requirements
- 11.10 Design
- 11.11 Source
- 11.12 Executable Object
- 11.13 Software Verification Cases and
- 11.14 Software Verification
- 11.15 Software Life Cycle Environment Configuration
- 11.16 Software Configuration
- 11.17 Problem
- 11.18 Software Configuration Management
- 11.19 Software Quality Assurance
- 11.20 Software Accomplishment

Section 12

ADDITIONAL CONSIDERATIONS

SECTION 12

- USE OF PREVIOUSLY DEVELOPED SOFTWARE (12.1)
- TOOL QUALIFICATION (12.2)
- ALTERNATIVE METHODS (12.3)

12.1 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Modiications to Previously Developed Software (12.1.1)
- Change of Aircraft Installation (12.1.2)
- Change of Application or Development Environment (12.1.3)
- Upgrading A Development Baseline (12.1.4)
- Software Configuration Management Considerations (12.1.5)
- Software Quality Assurance Considerations (12.1.6)

12.1 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Modiications to Previously Developed Software (12.1.1)
  - 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**(Upgrading A Development Baseline) 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(Software Verifications).

12.1 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Change of Aircraft Installation(12.1.2)
  - 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 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Change of Application or Development Environment (12.1.3)
  - a. If a new development environment **uses software development tools**, the guidelines of subsection **12.2**, Tool Qualification, may be.
  - b. The rigor of the evaluation of an application change should consider the **complexity and sophistication of the programming language**. For example, the rigor of the evaluation for Ada generics will be greater if the generic parameters are different in the new application. For object oriented languages, the rigor will be greater if the objects that are inherited are different in the new application.
  - c. If a different compiler or different set of compiler options are used, resulting in different object code, the results from a previous software verification process activity using the object code may **not be valid** and should not be used for the new application. In this case, previous test results may no longer be valid for the structural coverage criteria of the new application. Similarly, compiler assumptions about optimization may not be valid.

12.1 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Change of Application or Development Environment (12.1.3)
  - d. If a different processor is used then:

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

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

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

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

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

12.1 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Upgrading A Development Baseline (12.1.4)
  - 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 (USE OF PREVIOUSLY DEVELOPED SOFTWARE)

- Software Configuration Management Considerations (12.1.5)
  - 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.

- Software Quality Assurance Considerations (12.1.6)
  - a. Assurance that the software components satisfy or exceed the software life cycle criteria of the software level for the new application.
  - b. Assurance that changes to the software life cycle processes are stated on the software plans.

12.2 (TOOL QUALIFICATION)

- 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.

- 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.
  - **Software verification tools**: Tools that cannot introduce errors, but may fail to detect them.

12.2 (TOOL QUALIFICATION)

- 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.

12.2 (TOOL QUALIFICATION)

- Qualification Criteria for Software Development Tools(12.2.1)
- Qualification Criteria for Software Verification Tools(12.2.2)
- Tool Qualification Data(12.2.3)
  - Tool Qualification Plan(12.2.3.1)
  - Tool Operational Requirements(12.2.3.2)
- Tool Qualification Agreement(12.2.4)

12.2 (TOOL QUALIFICATION)

- Qualification Criteria for Software Development Tools(12.2.1)
  - 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.
  - 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.

12.2 (TOOL QUALIFICATION)

- Qualification Criteria for Software Development Tools(12.2.1)
  - 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 (TOOL QUALIFICATION)

- Qualification Criteria for Software Verification Tools(12.2.2)
- Tool Qualification Data(12.2.3)
  - 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 (TOOL QUALIFICATION)

- Tool Qualification Data(12.2.3) _Tool Qualification Plan(12.2.3.1)
  - 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.

- Tool Qualification Data(12.2.3) _ Tool Operational Requirements(12.2.3.2)
  - 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 (TOOL QUALIFICATION)

- Tool Qualification Agreement(12.2.4)
  - 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)

- 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 (ALTERNATIVE METHODS)

- Formal Methods(12.3.1)
- Exhaustive Input Testing(12.3.2)
- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3)
  - Independence of Multiple-Version Dissimilar Software(12.3.3.1)
  - Multiple Processor-Related Verification(12.3.3.2)
  - Multiple-Version Source Code Verification(12.3.3.3)
  - Tool Qualification for Multiple-Version Dissimilar Software(12.3.3.4)
  - Multiple Simulators and Verification(12.3.3.5)
- Software Reliability Models(12.3.4)
- Product Service History(12.3.5)

12.3 (ALTERNATIVE METHODS)

- Formal Methods(12.3.1)
  - Formal methods involve the use of formal logic, discrete mathematics, and computerreadable languages **to improve the specification and verification of software.**
  - 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
    - Coverage of software requirements and software architecture
    - Degree of rigor

12.3 (ALTERNATIVE METHODS)

- Exhaustive Input Testing(12.3.2)
  - 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 (ALTERNATIVE METHODS)

- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3)
  - 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.**

12.3 (ALTERNATIVE METHODS)

- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3)
  - 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.

12.3 (ALTERNATIVE METHODS)

- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3)
  - Independence of Multiple-Version Dissimilar Software(12.3.3.1)
  - Multiple Processor-Related Verification(12.3.3.2)
  - Multiple-Version Source Code Verification(12.3.3.3)
  - Tool Qualification for Multiple-Version Dissimilar Software(12.3.3.4)
  - Multiple Simulators and Verification(12.3.3.5)

12.3 (ALTERNATIVE METHODS)

- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3)

  _Independence of Multiple-Version Dissimilar Software(12.3.3.1)

  - 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.

- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3)

  _Multiple Processor-Related Verification(12.3.3.2)

  - 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 (ALTERNATIVE METHODS)

- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3) _Multiple-Version Source Code Verification(12.3.3.3)
  - a. Each version of software is coded using a different programming language.
  - b. Each compiler used is from a different developer.
- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3) _Tool Qualification for Multiple-Version Dissimilar Software(12.3.3.4)
  - a. Each tool was obtained from a different developer.
  - b. Each tool has a dissimilar design.
- Considerations for Multiple-Version Dissimilar Software Verification(12.3.3) _Multiple Simulators and Verification(12.3.3.5)
  - 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.

12.3 (ALTERNATIVE METHODS)

- Software Reliability Models(12.3.4)
  - 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 (ALTERNATIVE METHODS)

- Product Service History(12.3.5)
  - 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.

12.3 (ALTERNATIVE METHODS)

- 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.
- 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).

LEGEND

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).

DATA CONTROL CATEGORIES

SCMProcess Objective	Reference	CC1	CC2
Configuration Identification	7.2.1	o	o
Baselines	7.2.2 a, b, c, d, e	o	
Traceability	7.2.2f, g	o	o
Problem Reporting	7.2.3	o	
Change Control - integrity and identification	7.2.4a, b	o	o
Change Control - tracking	7.2.4c, d, e	o	
Change Review	7.2.5	o	
Configuration Status Accounting	7.2.6	o	
Retrieval	7.2.7a	o	o
Protection against Unauthorized Changes	7.2.7b(1)	o	o
Media Selection, Refreshing, Duplication	7.2.7b(2), (3), (4), c	o	
Release	7.2.7d	o	
Data Retention	7. 2.7e	o	o

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	

TABLE A-2
SOFTWARE DEVELOPMENT 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	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

TABLE A-2
SOFTWARE DEVELOPMENT PROCESS

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

- Software Requirements Data
  - A definition of the high-level requirements

a. Description of the allocation of system requirements to software.
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.
g. Failure detection and safety monitoring requirements.
h. Partitioning requirements allocated to software.

TABLE A-2
SOFTWARE DEVELOPMENT PROCESS

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

- DESIGN DESCRIPTION
  - a definition of the software architecture and the low-level requirements that will satisfy the software high-level requirements.

a. A detailed description of how the software satisfies the specified software high-level requirements.
b. The description of the software architecture defining the software structure to implement the requirements.
c. The input/output description.
d. The data flow and control flow of the design.
e. Resource limitations.
f. Scheduling procedures and inter-processor/inter-task communication mechanisms.
g. Design methods and details for their implementation.
h. Partitioning methods and means of preventing partition
i. Descriptions of the software components.
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.

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	

- SOFTWARE VERIFICATION RESULTS
  - produced by the software verification process activities.

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.

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 consistent.	6.3.3b	●	❍	❍		Software Verification Results	11.14	②	②	②	
10	Software architecture is compatible with target computer.	6.3.3c	❍	❍			Software Verification Results	11.14	②	②		
11	Software architecture is verifiable.	6.3.3d	❍	❍			Software Verification Results	11.14	②	②		
12	Software architecture conforms to standards.	6.3.3e	❍	❍	❍		Software Verification Results	11.14	②	②	②	
13	Software partitioning integrity is confirmed.	6.3.3f	●	❍	❍	❍	Software Verification Results	11.14	②	②	②	②

TABLE A-5
VERIFICATION OF OUTPUT SOFTWARE CODING AND 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	

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 Objet 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

- SOFTWARE VERIFICATION CASES AND PROCEDURES
  - detail how the software verification process activities are implemented.

  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.

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	

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 SCM Records	11.16 11.18	① ②	① ②	① ②	① 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 SCM Records	11.17 11.18	② ②	② ②	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 SCM Records	11.15 11.18	① ②	① ②	① ②	② ②

- 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.

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

- SOFTWARE CONFIGURATION INDEX
  - The Software Configuration Index (SCI) identifies the configuration of the software product.
  - 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.
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.

- PROBLEM REPORTS
  - 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.

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.

- 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.

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

- 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.

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 understand between the applicant and the certification authority is established.	9.0	❍	❍	❍	❍	Plan for Software Aspects of Certification	11.1	①	①	①	①
2	The means of compliance is proposed and agreement with the Plan for Software Aspect of Certification is obtained.	9.1	❍	❍	❍	❍	Plan for Software Aspect Certification	11.1	①	①	①	①
3	Compliance substantiation is provided.	9.2	❍	❍	❍	❍	Software Accomplishment Summary Software Configuration Index	11.20 11.16	① ①	① ①	① ①	① ①

- PLAN FOR SOFTWARE ASPECTS OF CERTIFICATION
  - 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.

  a. System overview
  b. Software overview
  c. Certification considerations
  d. Software life cycle
  e. Software life cycle data
  f. Schedule
  g. Additional considerations

- SOFTWARE ACCOMPLISHMENT SUMMARY
  - the primary data item for showing compliance with the Plan for Software Aspects of Certification.

a. System overview
b. Software overview
c. Certification considerations
d. Software characteristics
e. Software life cycle
f. Software life cycle data
g. Additional considerations
h. Software identification
i. Change history
j. Software status
k. Compliance statement