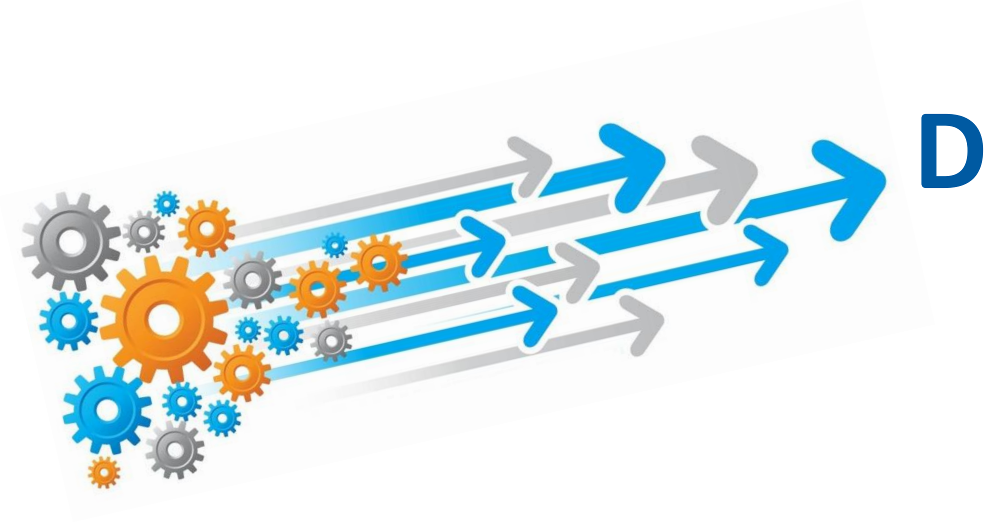


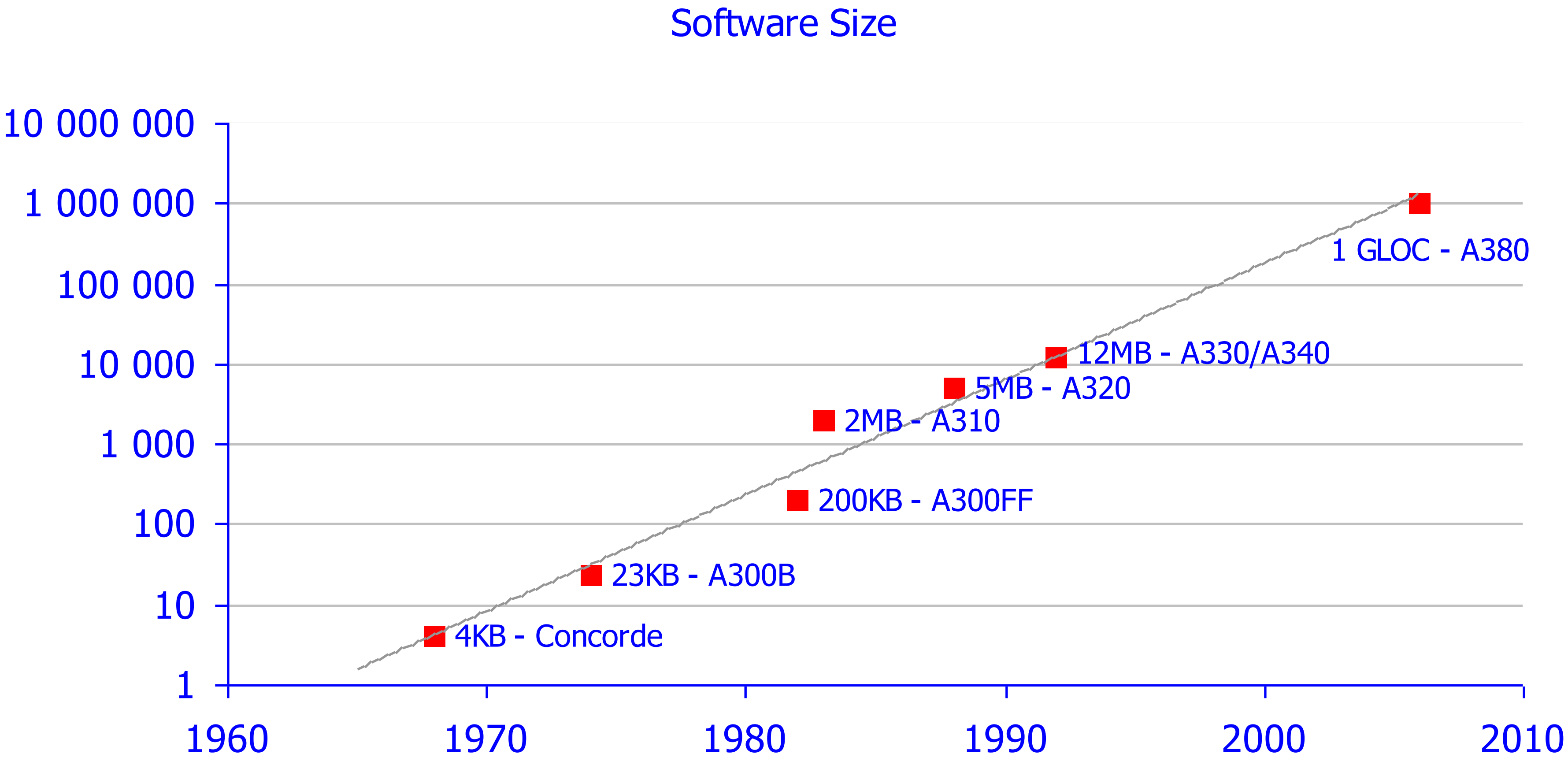
DO-178C / ED-12C Training

Mars 2023

Table of content

- **The evolution of the aeronautical world**
- Presentation of document
- Relation with other guides and level of criticality
- DO 178 Process
- Liaison with certification authorities

Increasing the size of embedded software



Increasing of embedded system

- **More than 50 critical embedded systems**
- **Dependent System (Connected via digital bus, ARINC-429, ARINC-629, AFDX)**
  - A lot of avionics functions :
    - Navigation, Flight Control, Auto-pilot, Stall Protection Systems,
    - Flight Warning System, Flight Instrumentation,
    - Landing Gear, Steering, Anti-skid Systems,
    - Fuel System, Engines Control System,
    - Air-ground Communications, …

Table of content

- The evolution of the aeronautical world
- **Presentation of document**
- Relation with other guides and level of criticality
- DO 178 Process
- Liaison with certification authorities

Why ? By who ?

- The rapid increase in the use of software in aeronautical equipment and systems has highlighted the need for a methodological handbook. This guide has been accepted by industries to meet the requirements of the aviation industry.
- D0-178 is the result of a consensus, reached during the long discussions of working groups.

  The document is published :

  - In Europe under the name ED-12C by EUROCAE
  - In USA under the name DO-178C by RTCA

ED-12C et DO-178C are strictly identical

For whom ? by whom?

- **D0-178** **Software Considerations in Airborne Systems and Equipment Certification :**
  - is aimed to product an embedded software with an aeronautical safety requirement.
  - is a set of objectives to be achieved through activities grouped into processes and recommendations.
  - is a set of considerations for the certification of on-board systems and equipment in order to obtain certificates of airworthiness.

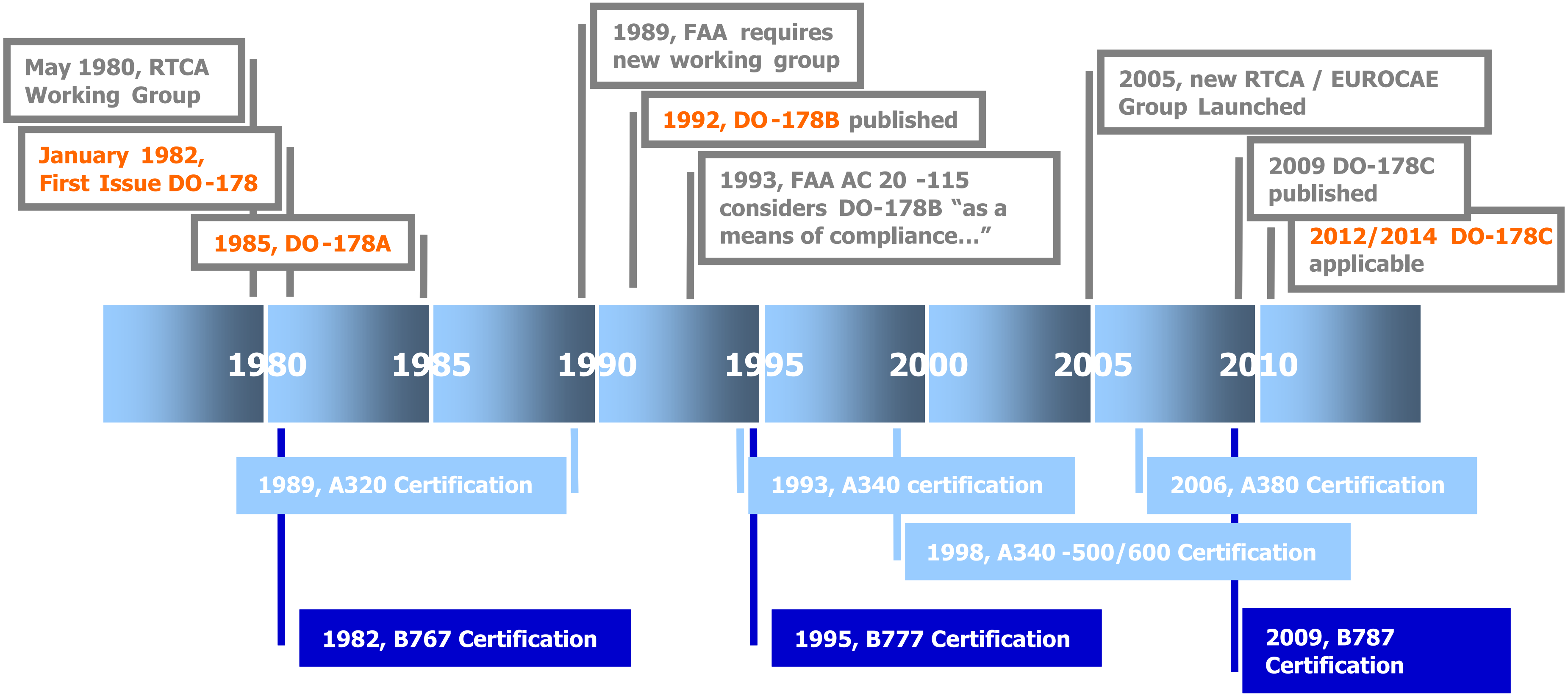
- **This standard express what is considered acceptable to safely embed software in aeronautical systems.**

DO-178 is primarily a document-oriented process

DO-178 and misleading

- **D0-178 do not advocate**:
  - Development life cycle
  - Technical limitation
- **Le D0-178 is not**:
  - A standard or a norm
  - A software architecture definition
  - Document template

Historic and key date



May 1980, RTCA Working Group

January 1982, First Issue DO-178

1985, DO-178A

1989, FAA requires new working group

1992, DO-178B published

1993, FAA AC 20 -115 considers DO-178B "as a means of compliance…"

2005, new RTCA / EUROCAE Group Launched

2009 DO-178C published

2012/2014 DO-178C applicable

1989, A320 Certification

1982, B767 Certification

1993, A340 certification

1998, A340 -500/600 Certification

1995, B777 Certification

2006, A380 Certification

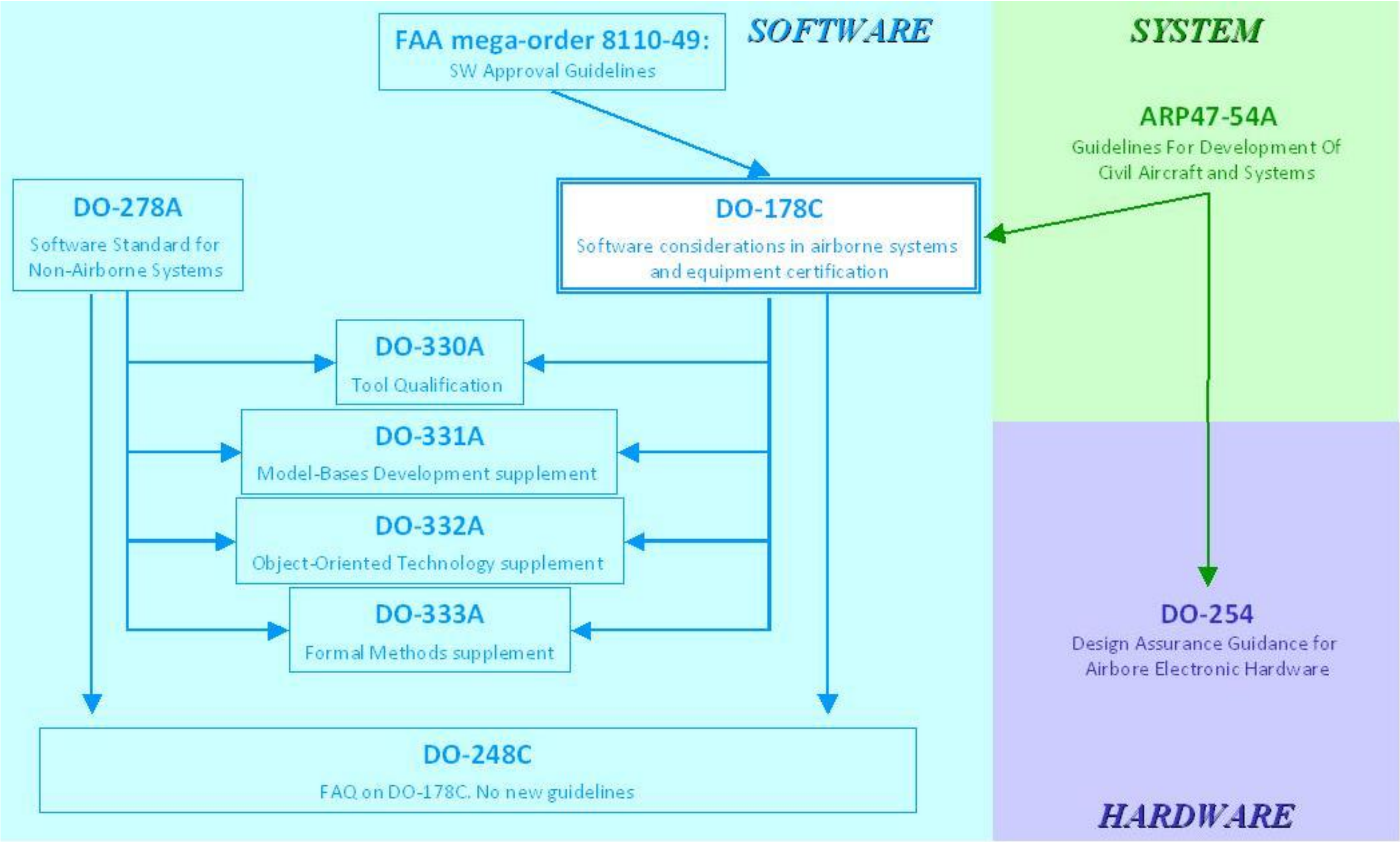
2009, B787 Certification

Table of content

- The evolution of the aeronautical world
- Presentation of document
- **Relation with other guides and level of criticality**
- DO 178 Process
- Liaison with certification authorities

System certification

- **The** **safety** **objectives are passed from the system level to all lower levels**



SOFTWARE
SYSTEM
FAA mega-order 8110-49:
SW Approval Guidelines
ARP47-54A
Guidelines For Development Of Civil Aircraft and Systems
DO-278A
Software Standard for Non-Airborne Systems
DO-178C
Software considerations in airborne systems and equipment certification
DO-330A
Tool Qualification
DO-331A
Model-Bases Development supplement
DO-332A
Object-Oriented Technology supplement
DO-333A
Formal Methods supplement
DO-254
Design Assurance Guidance for Airbore Electronic Hardware
DO-248C
FAQ on DO-178C. No new guidelines
HARDWARE

DO-178 purpose

- **The purpose of the recommendations given by DO-178 is the minimizing the number and severity of error in the software, because it is impossible a posteriori**
  - To check that a software fully meets the need
  - Measure or evaluate its quality and fiability

The recommendations are therefore focused, through the life cycle processes of a software :

- To the prevention of errors
- Detection and removal of errors

DO-178 consideration

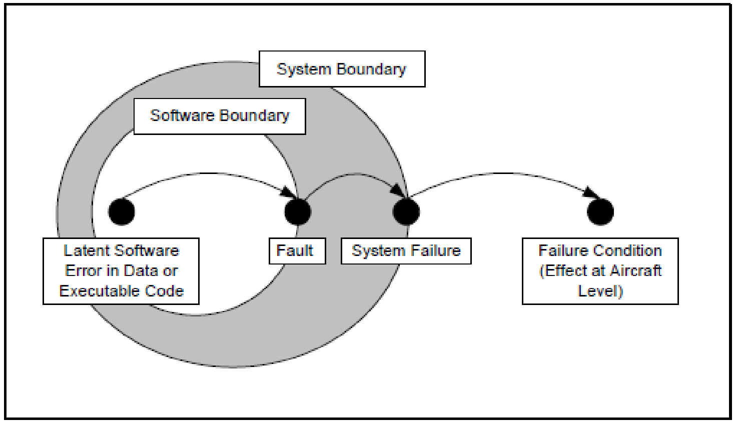
- **Set objectives to reach (§Annexe A) via activities grouped into processes and recommendations to meet the aeronautical safety requirements**
  - Is not prescriptive on the form of processes, nor a development methodology nor in the structure of documents
- **Defined a minimal content of documents (§11)**
- **Defined a communication between certification authorities and clarify the relationship between the system and the software (§1)**
- **Demands proofs that within the processes the activities were carried out and that the objectives were achieved → a maximun of formalism is required**
  - More the target quality is highter, more proof to provide
  - Evidence proof is expensive

Design Assurance Level (DAL) definition

- **DAL = Design Assurance Level**
  - DAL should be determined for both quantitative and qualitative aspects based on the severity of the consequences of software errors on the system and the consideration of equipment architecture considerations..
- **Classification of criticality**
  - Criticality is related to the severity of the effects of the associated failure condition. It also depends on the architecture of the system, in particular the number of external events that can cause a failure.
  - The condition of failure / failure s the effect on the airplane and its occupants, both directly

  or as a result, caused by one or

  more failures/



System Boundary
Software Boundary
Latent Software Error in Data or Executable Code
Fault
System Failure
Failure Condition (Effect at Aircraft Level)

Failures classification

- **Failure conditions are classified according to their impact at the system level :**
  - Catastrophic : conditions due to multiple sources of fatal errors that most often lead to the loss of the airplane.

    ➔ **DAL A**
  - Hazardous : Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers. (Safety-significant)

    ➔ **DAL B**
  - Major : Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries)

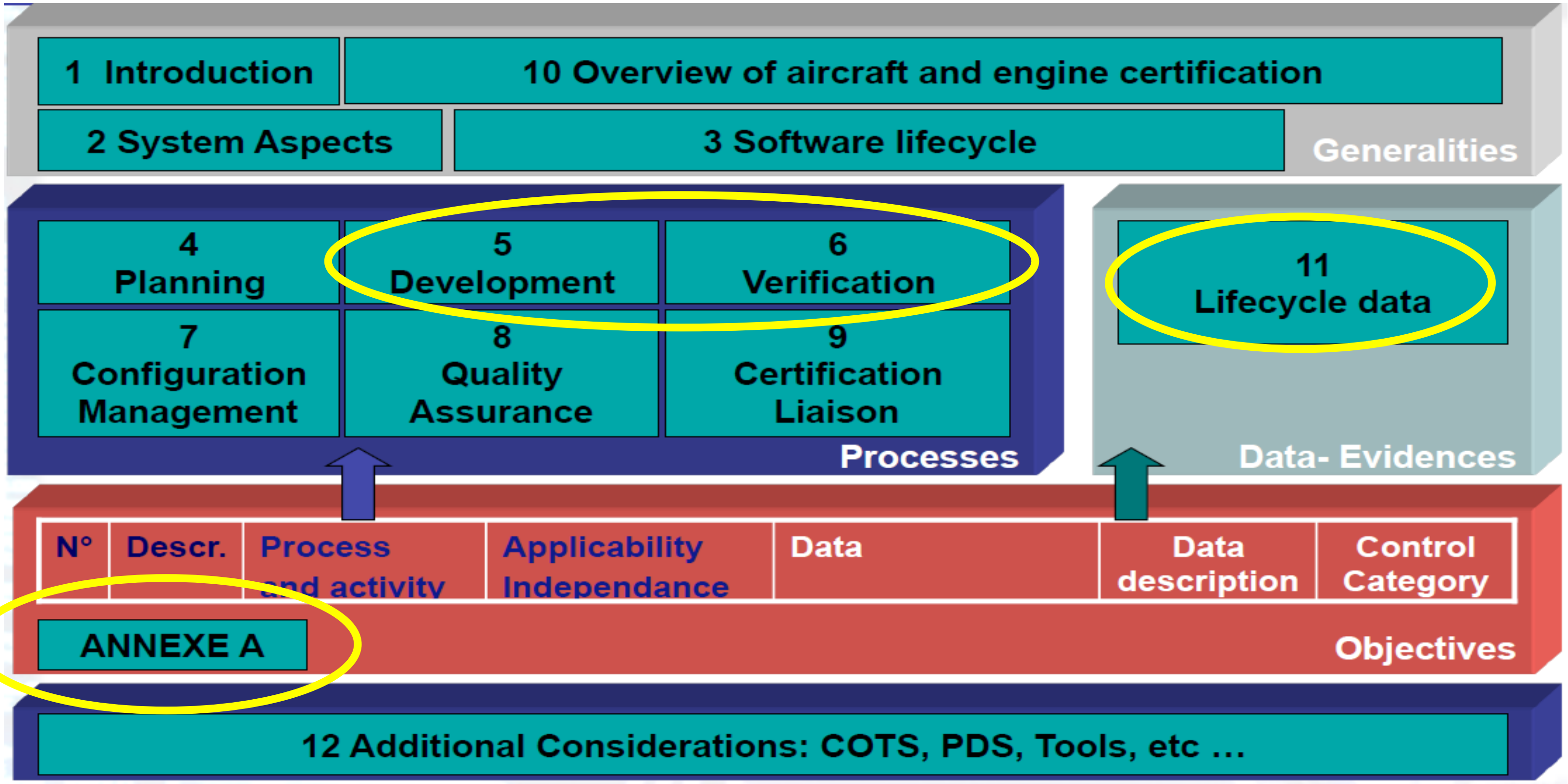
    ➔ **DAL C**
  - Minor : That do not reduce flight safety.

    ➔ **DAL D**

Table of content

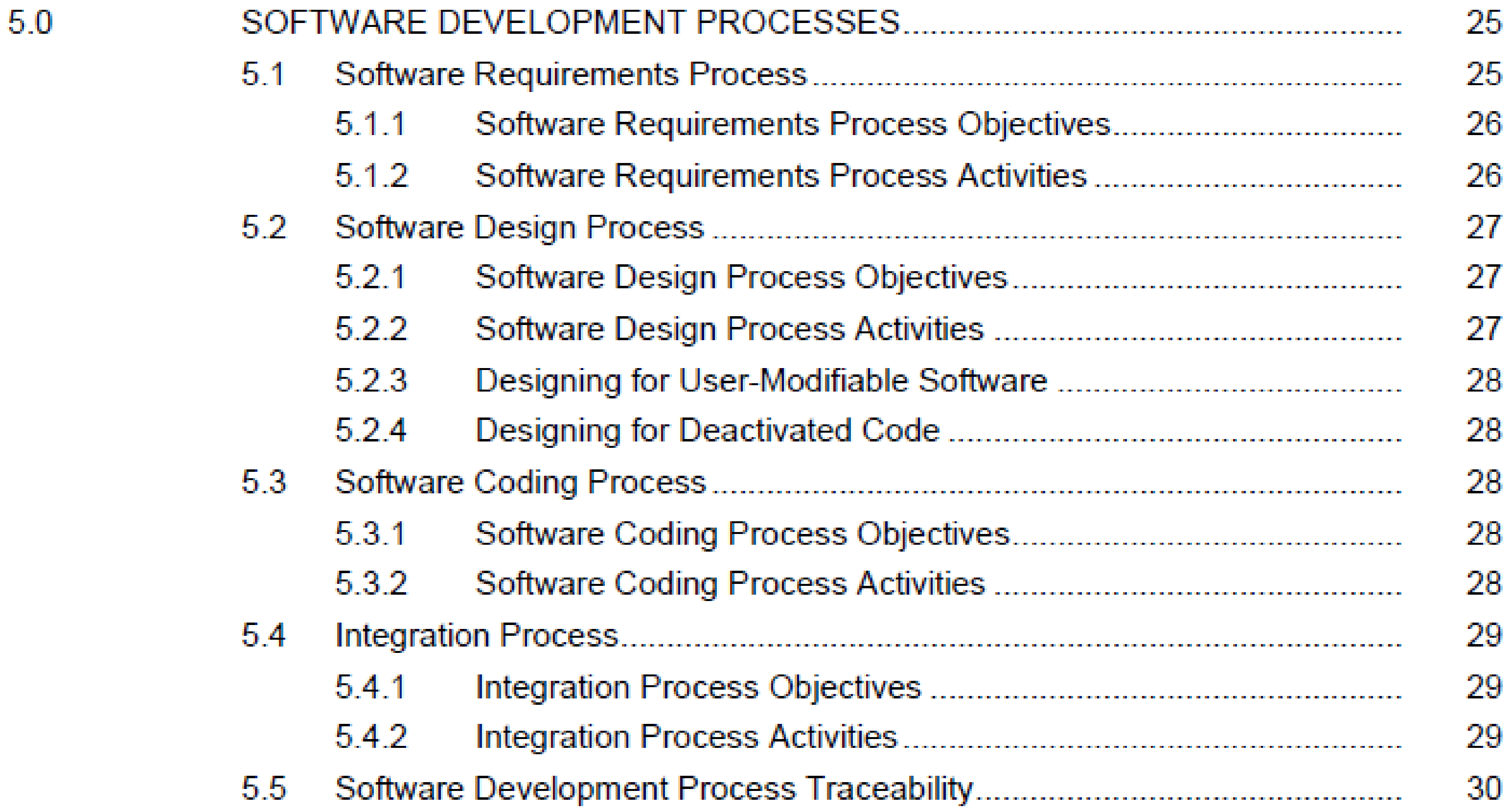
- The evolution of the aeronautical world
- Presentation of document
- Relation with other guides and level of criticality
- **DO 178 Process**
- Liaison with certification authorities

Summary of DO-178



- Document overview in 12 chapters
- Annex A: 10 summary tables of objectives to be achieved and data to be produced during development
- Annex B: Acronyms and definition of terms used

5 -Development processes - overview



5.0 SOFTWARE DEVELOPMENT PROCESSES 25

5.1 Software Requirements Process 25

5.1.1 Software Requirements Process Objectives 26

5.1.2 Software Requirements Process Activities 26

5.2 Software Design Process 27

5.2.1 Software Design Process Objectives 27

5.2.2 Software Design Process Activities 27

5.2.3 Designing for User-Modifiable Software 28

5.2.4 Designing for Deactivated Code 28

5.3 Software Coding Process 28

5.3.1 Software Coding Process Objectives 28

5.3.2 Software Coding Process Activities 28

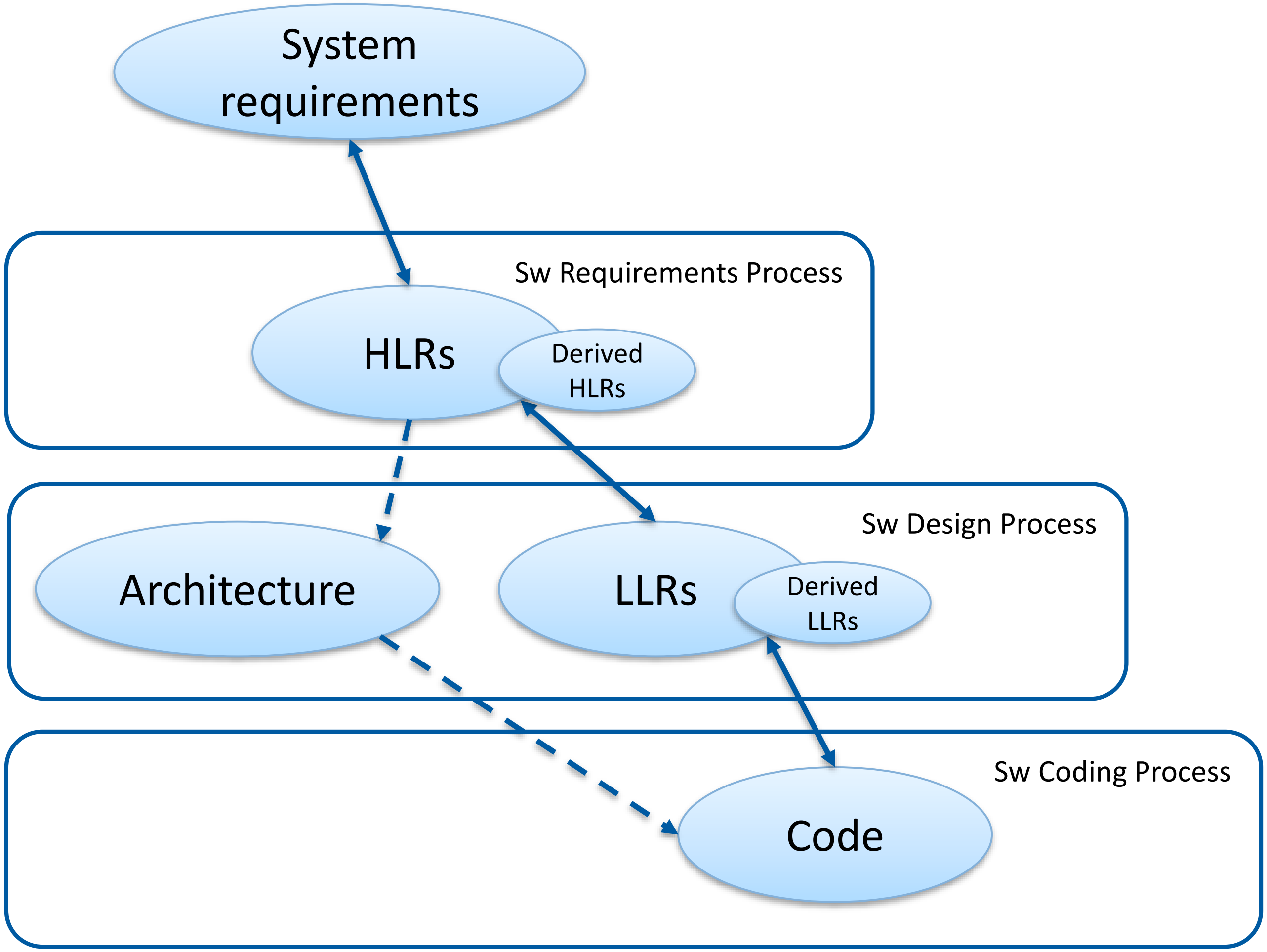
5.4 Integration Process 29

5.4.1 Integration Process Objectives 29

5.4.2 Integration Process Activities 29

5.5 Software Development Process Traceability 30

5 -Development processes - overview



⟷ bidirectional traceability links

4 -Development processes - Requirements level

- HLR: **High-level requirements** are produced directly through analysis of system requirements and system architecture

- LLR: **Low-level requirements** are software requirements from which Source Code can be directly implemented without further information. LLR must are directly traceable to **HLR.**

- **Derived requirements** are requirements that are not directly traceable to higher level requirements
  - Example: interrupt handling software for the chosen target computer

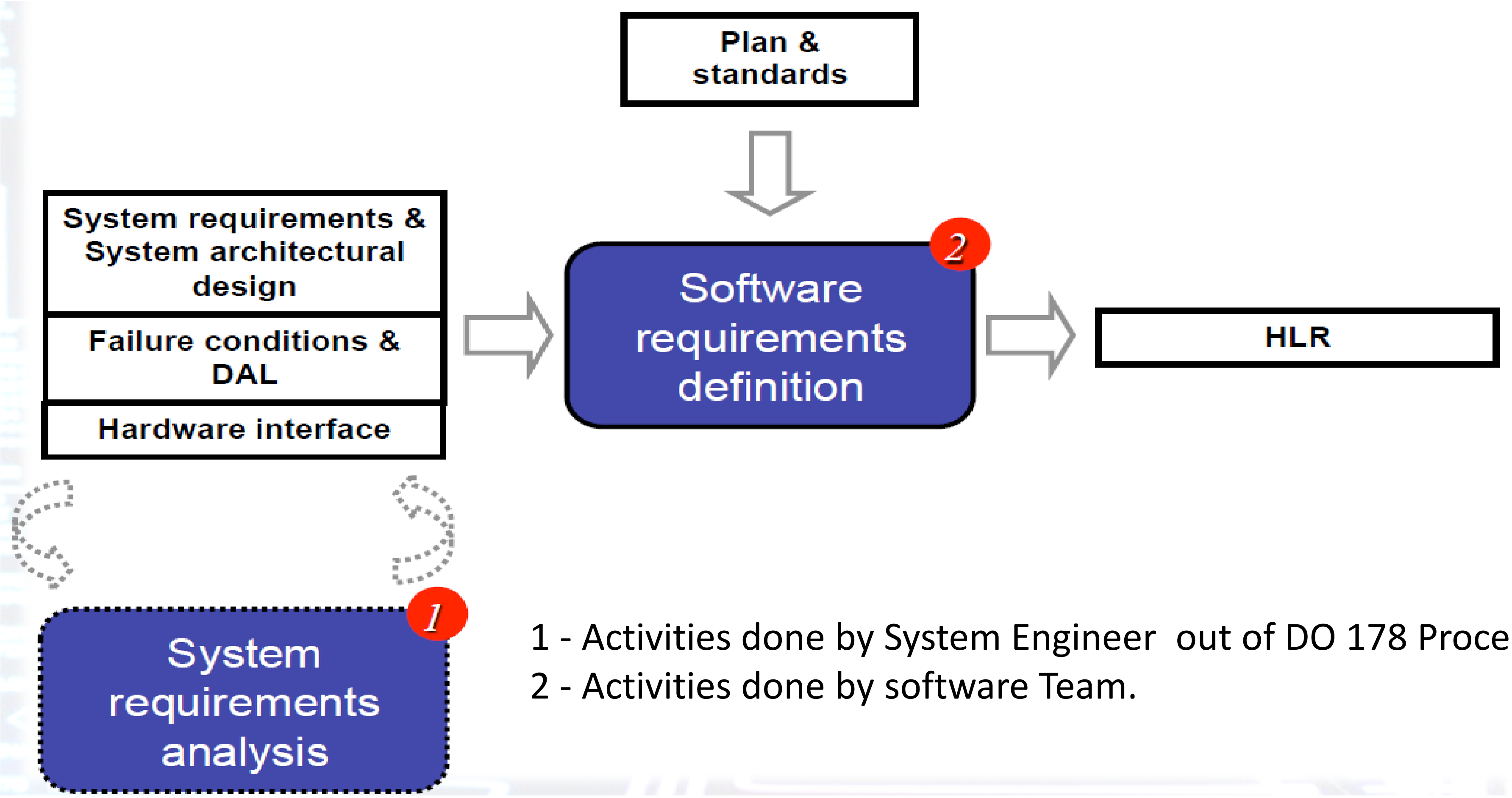
High-Level Requirements and Low-Level Requirements may include derived requirements

4 -Development processes – High Level Requirement

- **Objectives**:
  - High-level requirements are **developed**
  - Derived high-level requirements are indicated to the system safety assessment process

- **HLR** :
  - “**Why/what**”, in opposition to “How”
  - Functional requirements describe the **functions** that the software is **to execute**
  - Non-functional requirements are the ones that act to constrain the solution (performance, maintainability, safety, reliability…)

4 -Development processes – High Level Requirement


Plan & standards
System requirements & System architectural design
Failure conditions & DAL
Hardware interface
2
Software requirements definition
HLR
1
System requirements analysis

1 - Activities done by System Engineer out of DO 178 Process

2 - Activities done by software Team.

4 -Development processes – High Level Requirement

- The system functional and interface requirements that are allocated to software should not have **ambiguities**, **inconsistencies** and undefined **conditions**
- Each system requirement should be specified in HLR
- HLR should conform to the Software Requirements Standards, and be verifiable and consistent
- HLR should **be stated in quantitative terms** with tolerances applicable
- HLR should not describe design or verification detail except for specified and justified design constraints
- Derived HLR should be provided to the system safety assessment process

4 -Development processes – High Level Requirement

HLR plus derived should include:

- Description of the allocation of system requirements to software, with attention to safety-related requirements and potential failure conditions
- Functional and operational requirements under each mode of operation Performance criteria, for example, precision and accuracy
- **Timing requirements** and constraints
- **Memory** size constraints
- **Hardware and software interfaces**, Example: protocols, formats frequency inputs/outputs
- Failure detection and safety monitoring requirements
- **Partitioning requirements** allocated to software, how the partitioned software components

4 -Development processes – Design Process

- **Objectives**:
  - The software architecture and low-level requirements are developed from the HLR
  - Derived low-level requirements are provided to the system safety assessment process

- **Architecture** :
  - Belongs to the field of the solutions (“How”)
  - **Architectural design** is the transformation of HLR to the software item into an architecture that describes its top-level structure and identifies the software components
  - All **HLR** are allocated to its software components and further **refined fo detailed design**
  - The developer develops and documents a top-level design for interfaces between the software components of the software item
  - For Real-time systems, the developer develops dynamic architecture that describes the functions allocation to processes/threads/tasks, their scheduling, the means of communication between them, the exception/interruption handling

4 -Development processes – Design Process

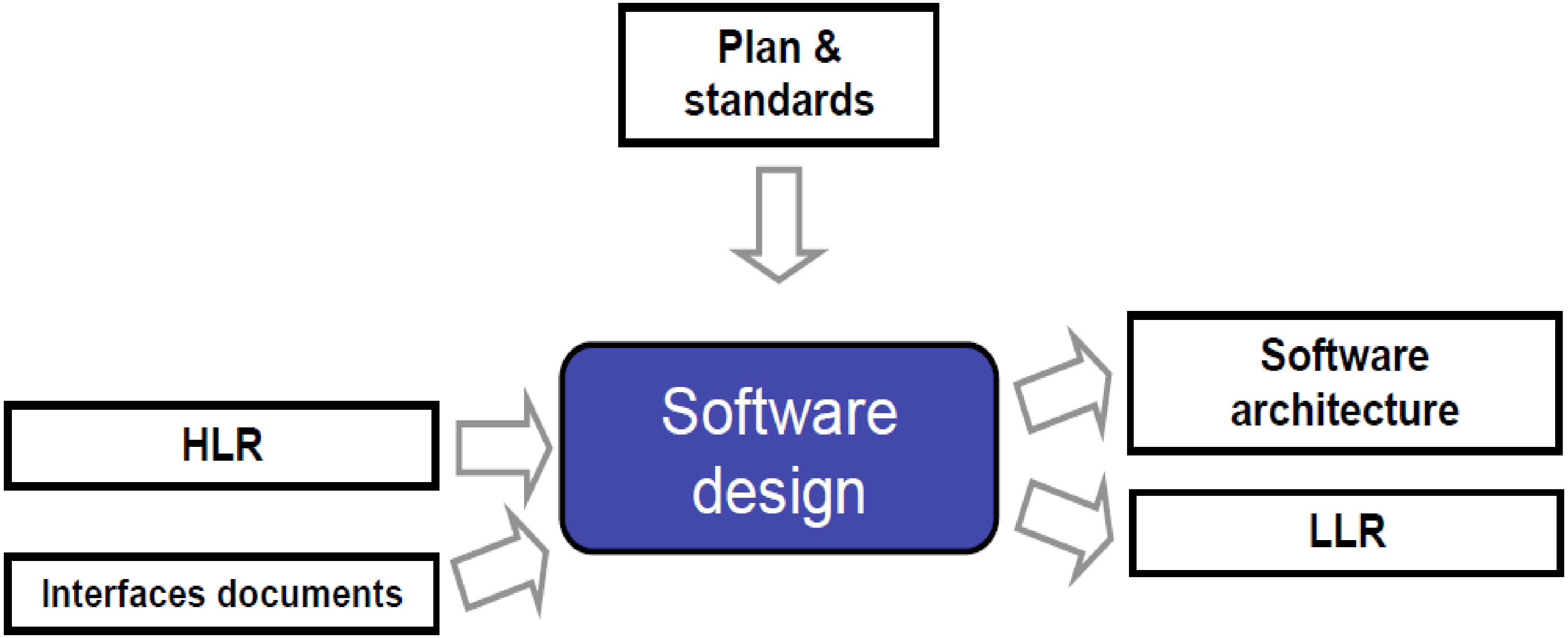
- **LLR:**
  - The design refines each component identified in the architectural design into lower levels containing software units. The detailed design shall be documented.
  - Main methods for detailed design:
    - Function-oriented, also called structured (focused on functions, refined in a top-down
    - Data-Structure centered (focused on inputs/outputs)
    - Object-oriented (data abstraction, inheritance and polymorphism)
  - Detailled design should define:
    - Type of Data
    - External or internal functions
    - Prototype of function
    - Usability of Data (private our public)

4 -Development processes – Design Process

- **Architecture + LLR + derived LLR should include:**

- A detailed description of how the software satisfies the specified HLR, including algorithms, data structures, and how software requirements are allocated to processors and tasks
- The description of the software architecture defining the software structure to implement the requirements
- The input/output description, for example, a data dictionary, both internally and externally throughout the software architecture
- The data flow and control flow of the design
- Resource limitations, the strategy for managing each resource and its limitations Scheduling procedures and inter-processor/inter-task communication mechanisms, including time-rigid sequencing, preemptive scheduling, Ada rendezvous, and interrupts

4 -Development processes – Design Process



4 -Development processes – Coding Process

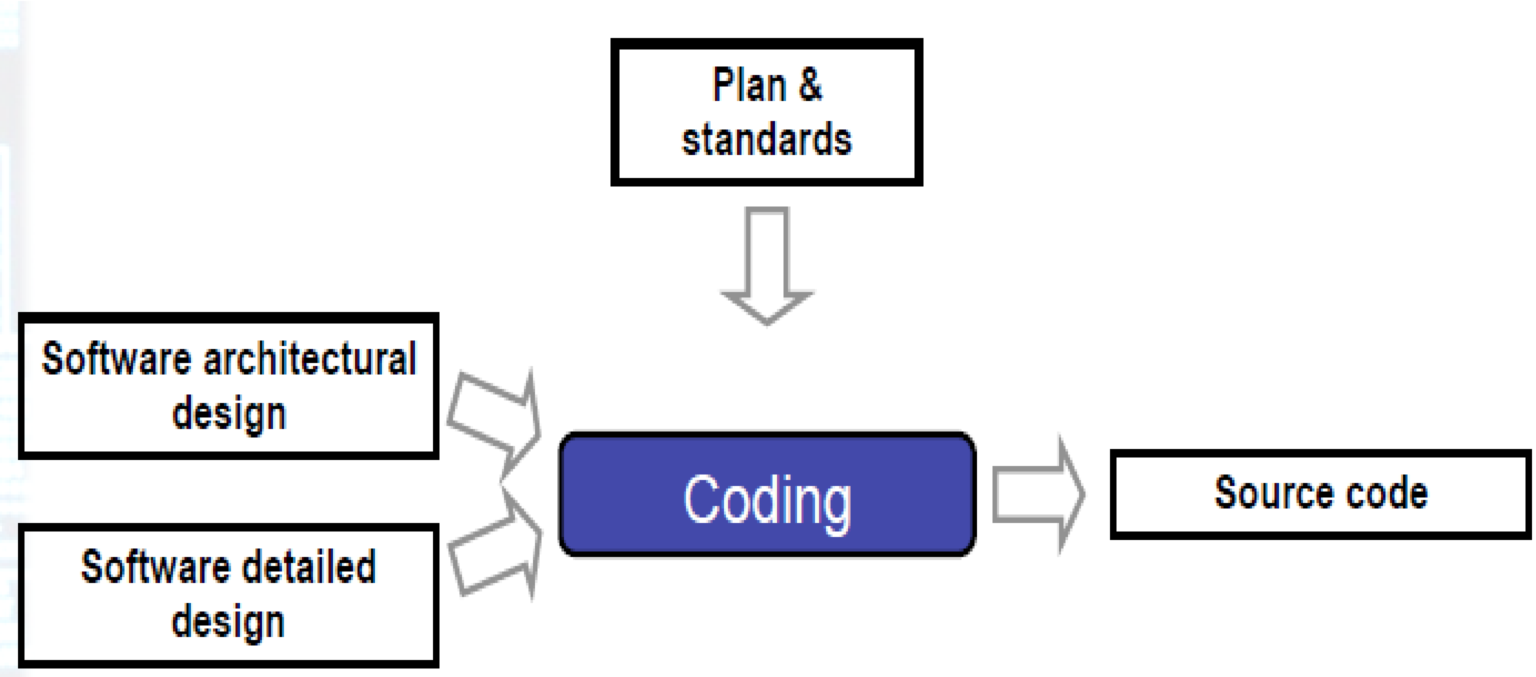
Objective:

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

Activities:

- The Source Code should implement LLR and conform to the software architecture
- The Source Code should conform to the Software Code Standards
- The Source Code should be traceable to the Design Description

4 -Development processes – Coding Process



4 -Development processes – Coding Process

Coding data:

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

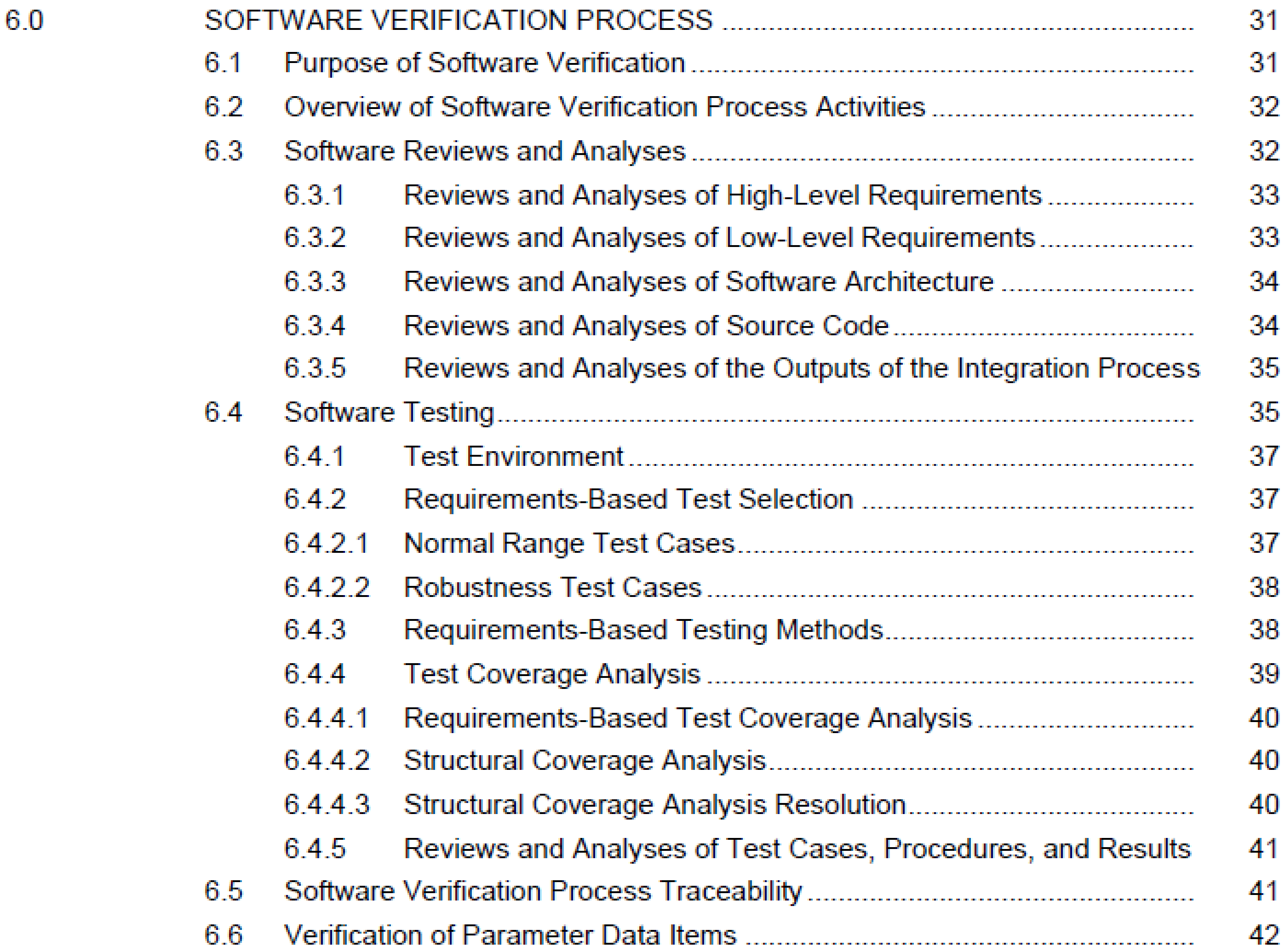
4 -Development processes – Coding Process

Coding standards should include:

- **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
- **Source Code presentation standards**, for example, line length restriction, indentation, and blank line usage and Source Code documentation standards, author, revision history, inputs and outputs
- **Naming conventions** for components, subprograms, variables, and constants

- **Conditions and constraints imposed** on permitted coding conventions, such as the degree of coupling between software components, the nesting for control structures and the complexity of logical or numerical expressions and rationale for their use
- Constraints on the use of the coding tools

6 – Verification process – Overview



6.0 SOFTWARE VERIFICATION PROCESS 31

6.1 Purpose of Software Verification 31

6.2 Overview of Software Verification Process Activities 32

6.3 Software Reviews and Analyses 32

6.3.1 Reviews and Analyses of High-Level Requirements 33

6.3.2 Reviews and Analyses of Low-Level Requirements 33

6.3.3 Reviews and Analyses of Software Architecture 34

6.3.4 Reviews and Analyses of Source Code 34

6.3.5 Reviews and Analyses of the Outputs of the Integration Process 35

6.4 Software Testing 35

6.4.1 Test Environment 37

6.4.2 Requirements-Based Test Selection 37

6.4.2.1 Normal Range Test Cases 37

6.4.2.2 Robustness Test Cases 38

6.4.3 Requirements-Based Testing Methods 38

6.4.4 Test Coverage Analysis 39

6.4.4.1 Requirements-Based Test Coverage Analysis 40

6.4.4.2 Structural Coverage Analysis 40

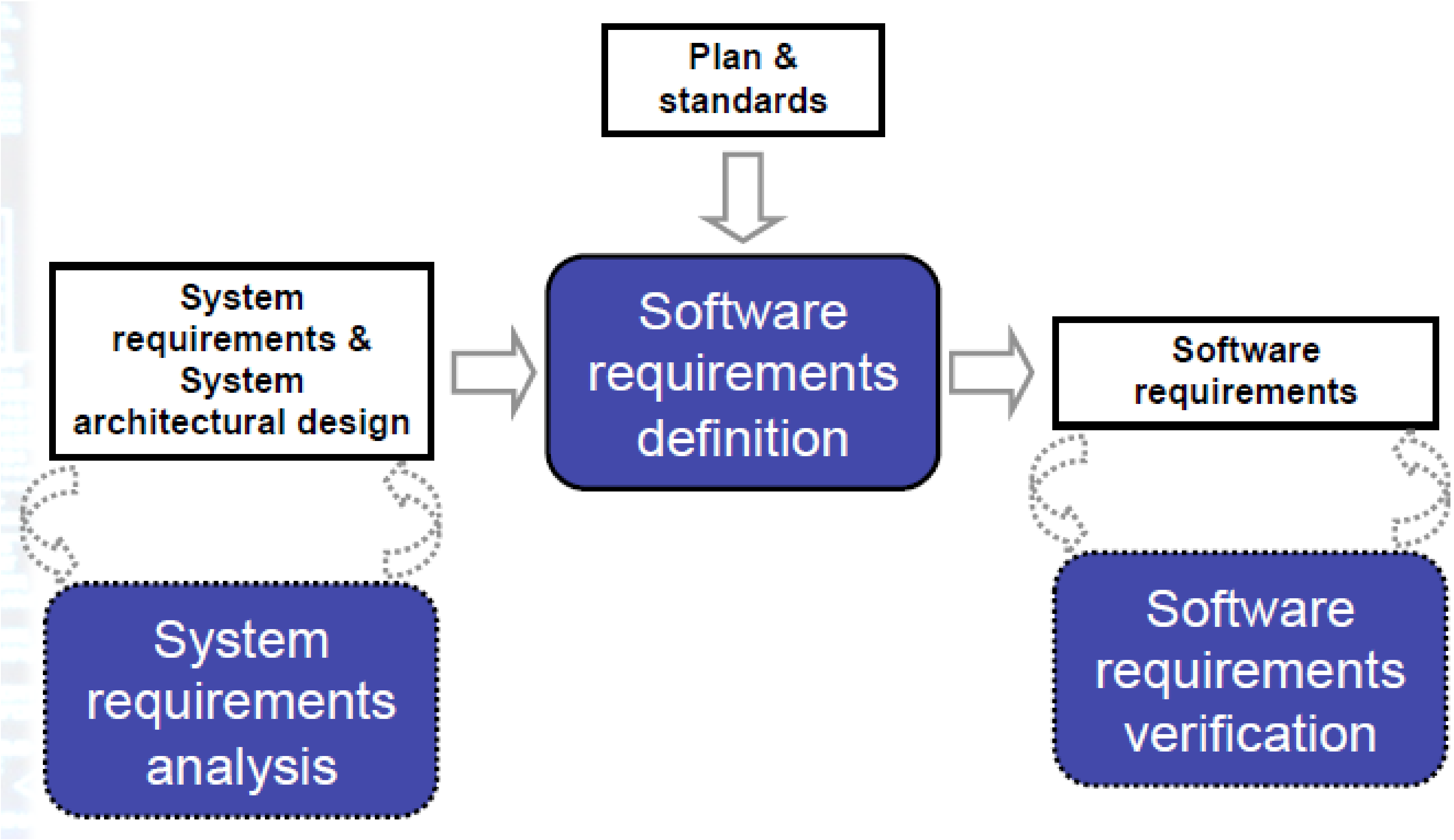
6.4.4.3 Structural Coverage Analysis Resolution 40

6.4.5 Reviews and Analyses of Test Cases, Procedures, and Results 41

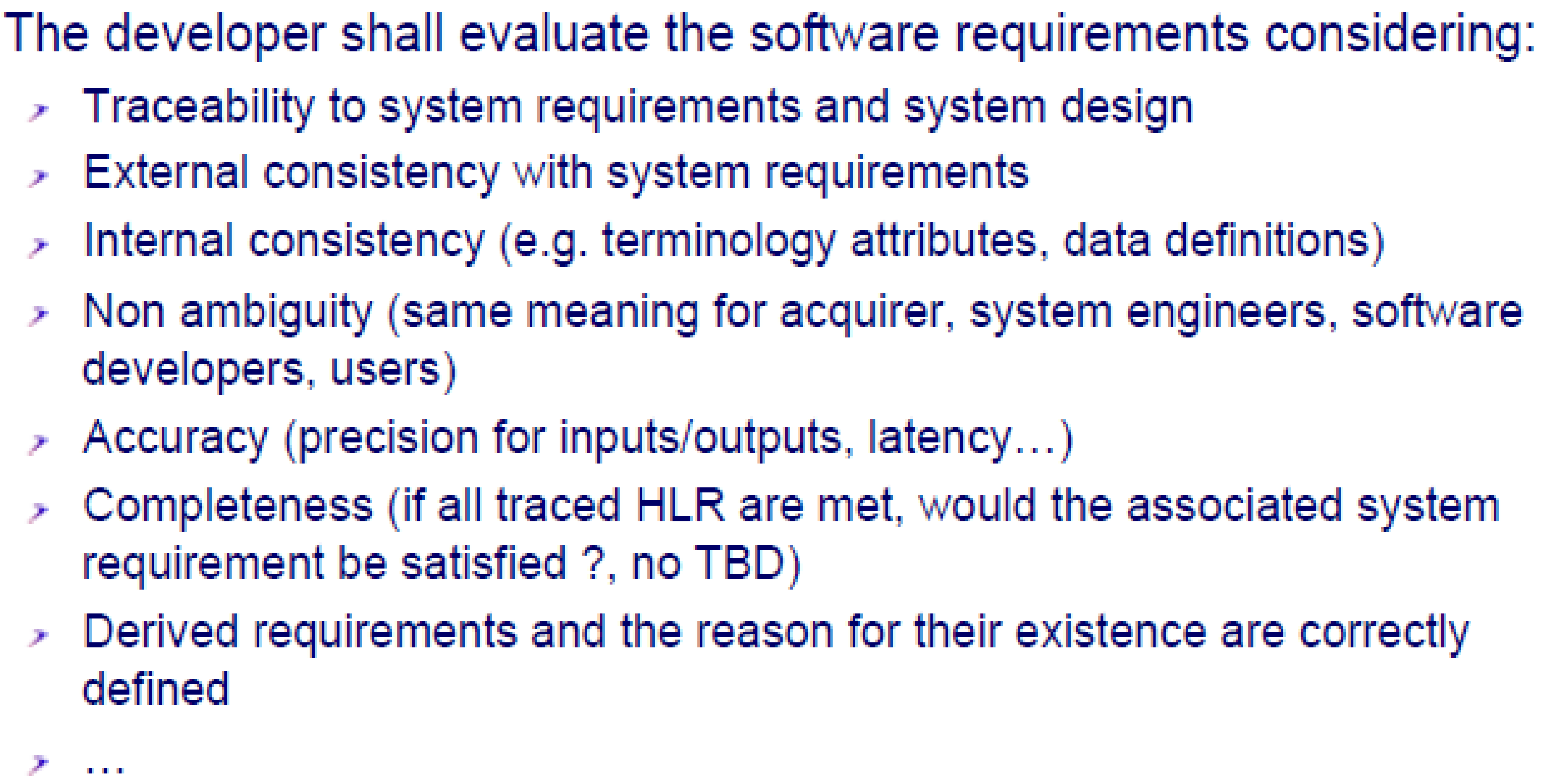
6.5 Software Verification Process Traceability 41

6.6 Verification of Parameter Data Items 42

6 – Verification process – requirement Verification


Plan & standards
System requirements & System architectural design
Software requirements definition
Software requirements
System requirements analysis
Software requirements verification

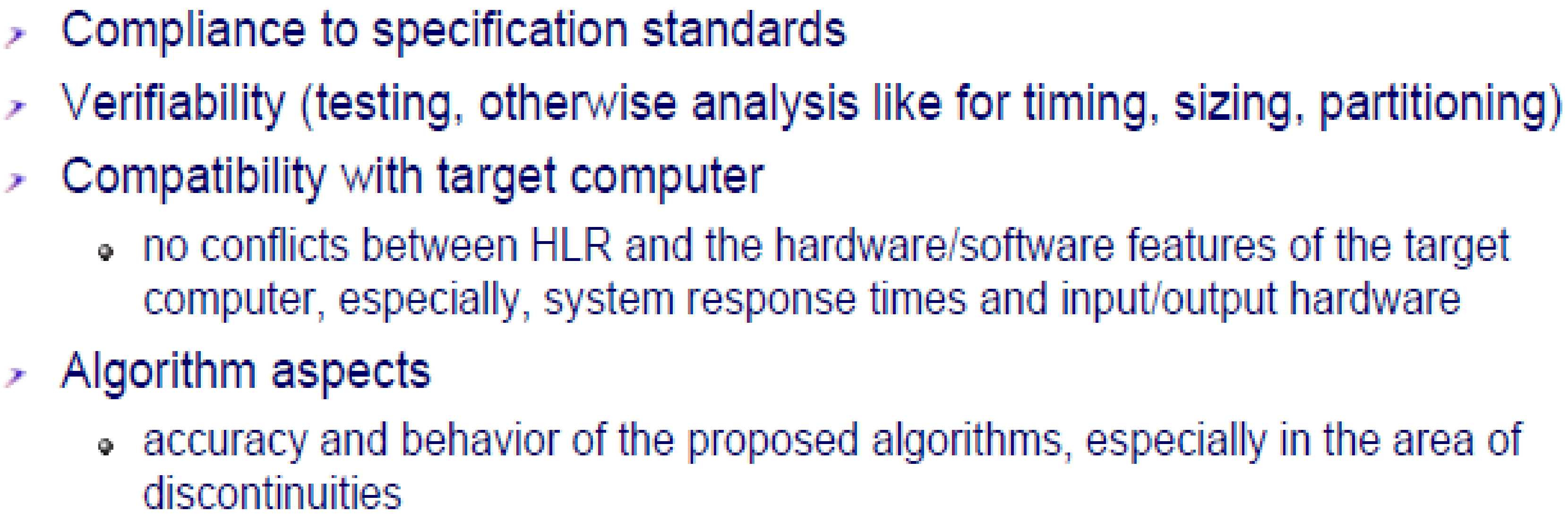
6 – Verification process – HLR Verification



The developer shall evaluate the software requirements considering:

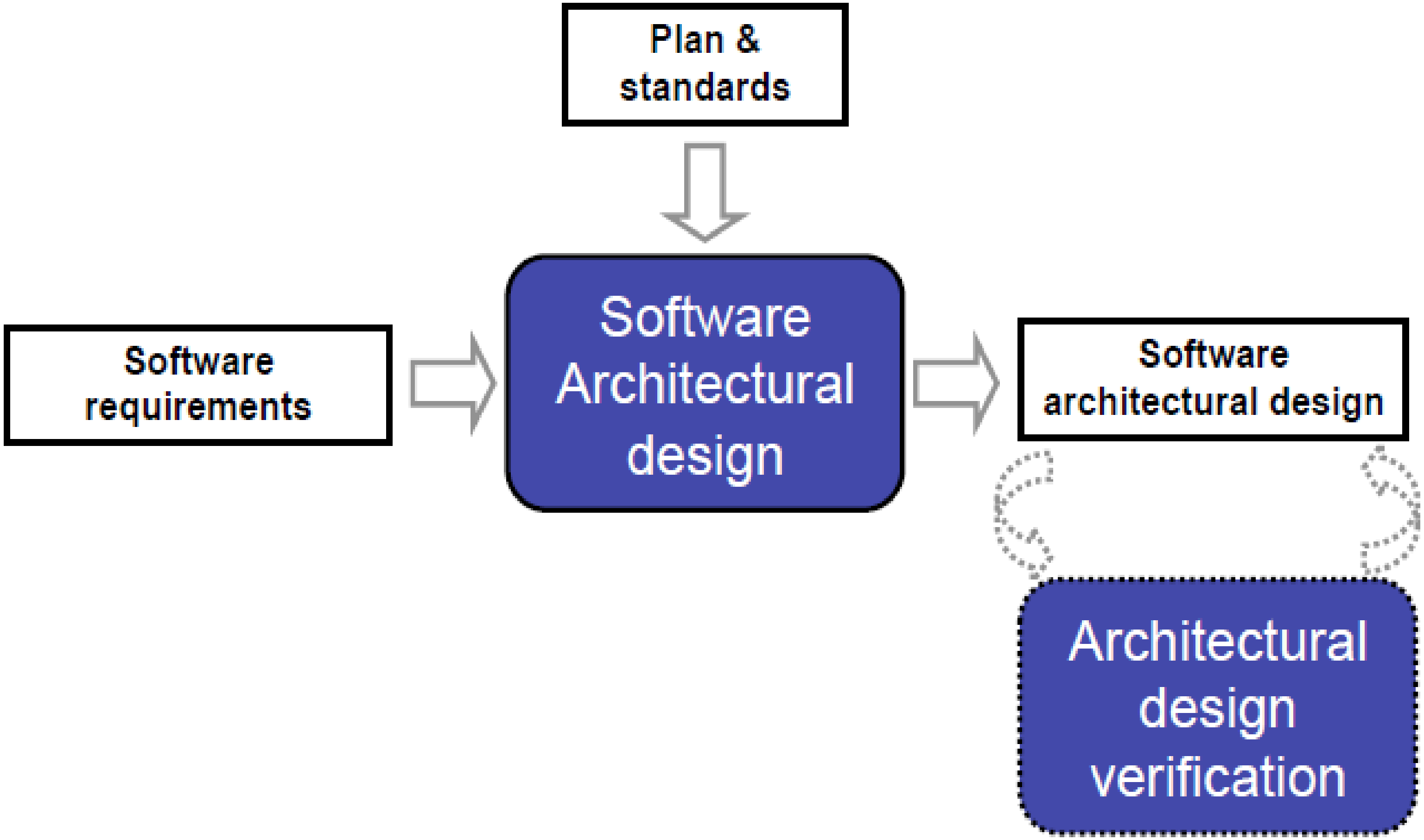
- Traceability to system requirements and system design
- External consistency with system requirements
- Internal consistency (e.g. terminology attributes, data definitions)
- Non ambiguity (same meaning for acquirer, system engineers, software developers, users)
- Accuracy (precision for inputs/outputs, latency…)
- Completeness (if all traced HLR are met, would the associated system requirement be satisfied ?, no TBD)
- Derived requirements and the reason for their existence are correctly defined
- …

6 – Verification process – HLR Verification

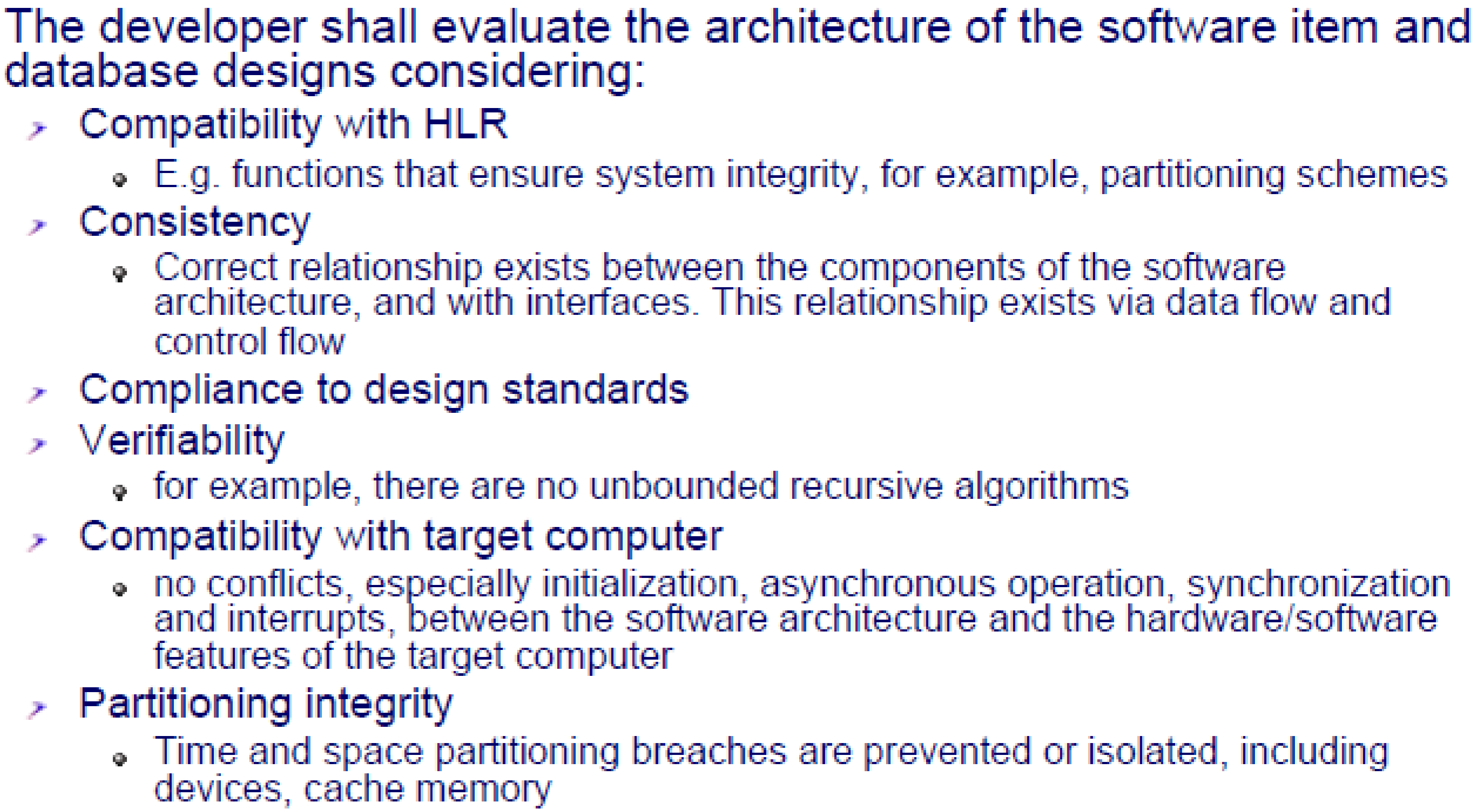


- Compliance to specification standards
- Verifiability (testing, otherwise analysis like for timing, sizing, partitioning)
- Compatibility with target computer
  - no conflicts between HLR and the hardware/software features of the target computer, especially, system response times and input/output hardware
- Algorithm aspects
  - accuracy and behavior of the proposed algorithms, especially in the area of discontinuities

6 – Verification process – Architecture Verification


Plan & standards
Software requirements
Software Architectural design
Software architectural design
Architectural design verification

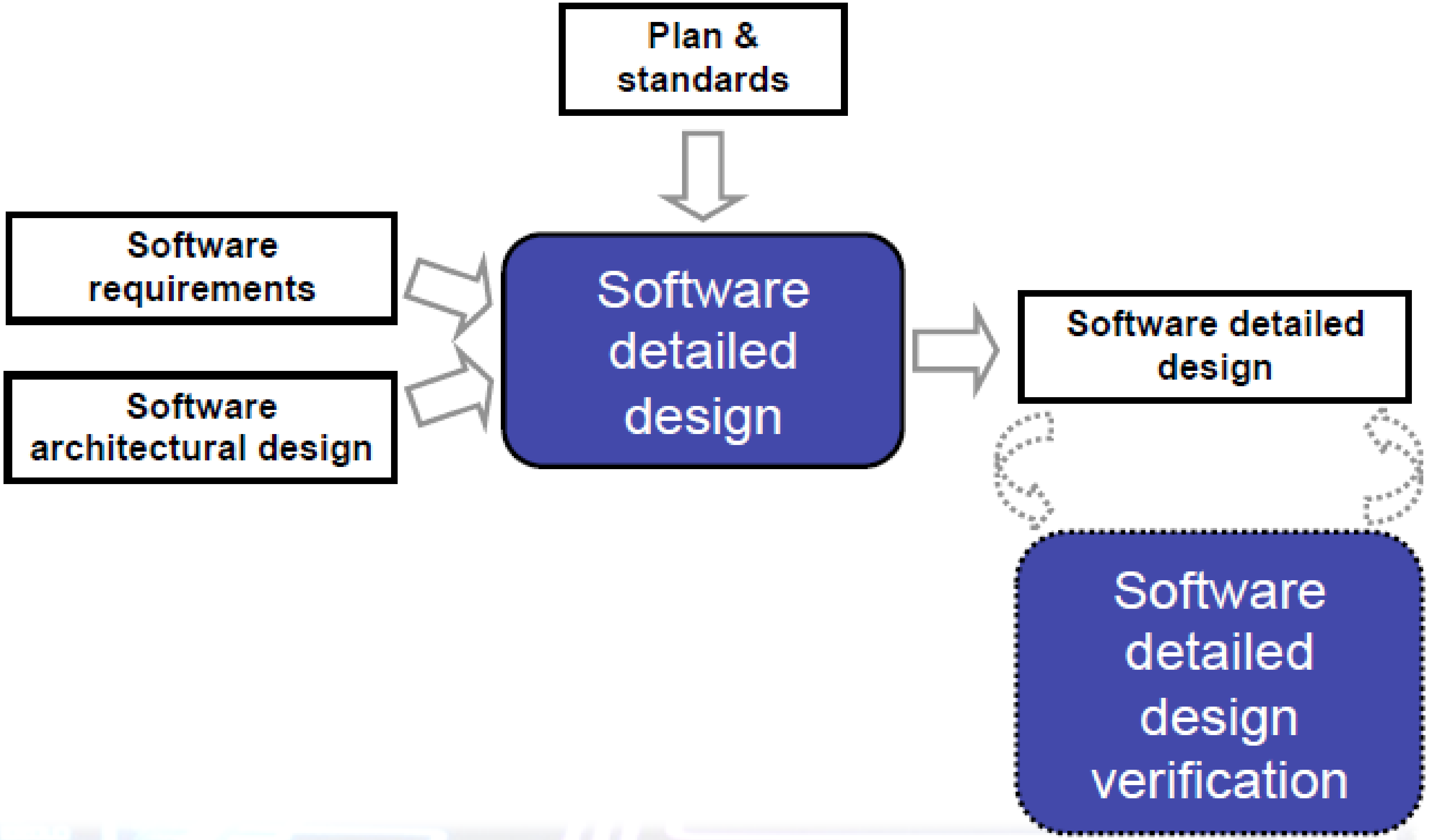
6 – Verification process – Architecture Verification



The developer shall evaluate the architecture of the software item and database designs considering:

- Compatibility with HLR
  - E.g. functions that ensure system integrity, for example, partitioning schemes
- Consistency
  - Correct relationship exists between the components of the software architecture, and with interfaces. This relationship exists via data flow and control flow
- Compliance to design standards
- Verifiability
  - for example, there are no unbounded recursive algorithms
- Compatibility with target computer
  - no conflicts, especially initialization, asynchronous operation, synchronization and interrupts, between the software architecture and the hardware/software features of the target computer
- Partitioning integrity
  - Time and space partitioning breaches are prevented or isolated, including devices, cache memory

6 – Verification process – LLR Verification

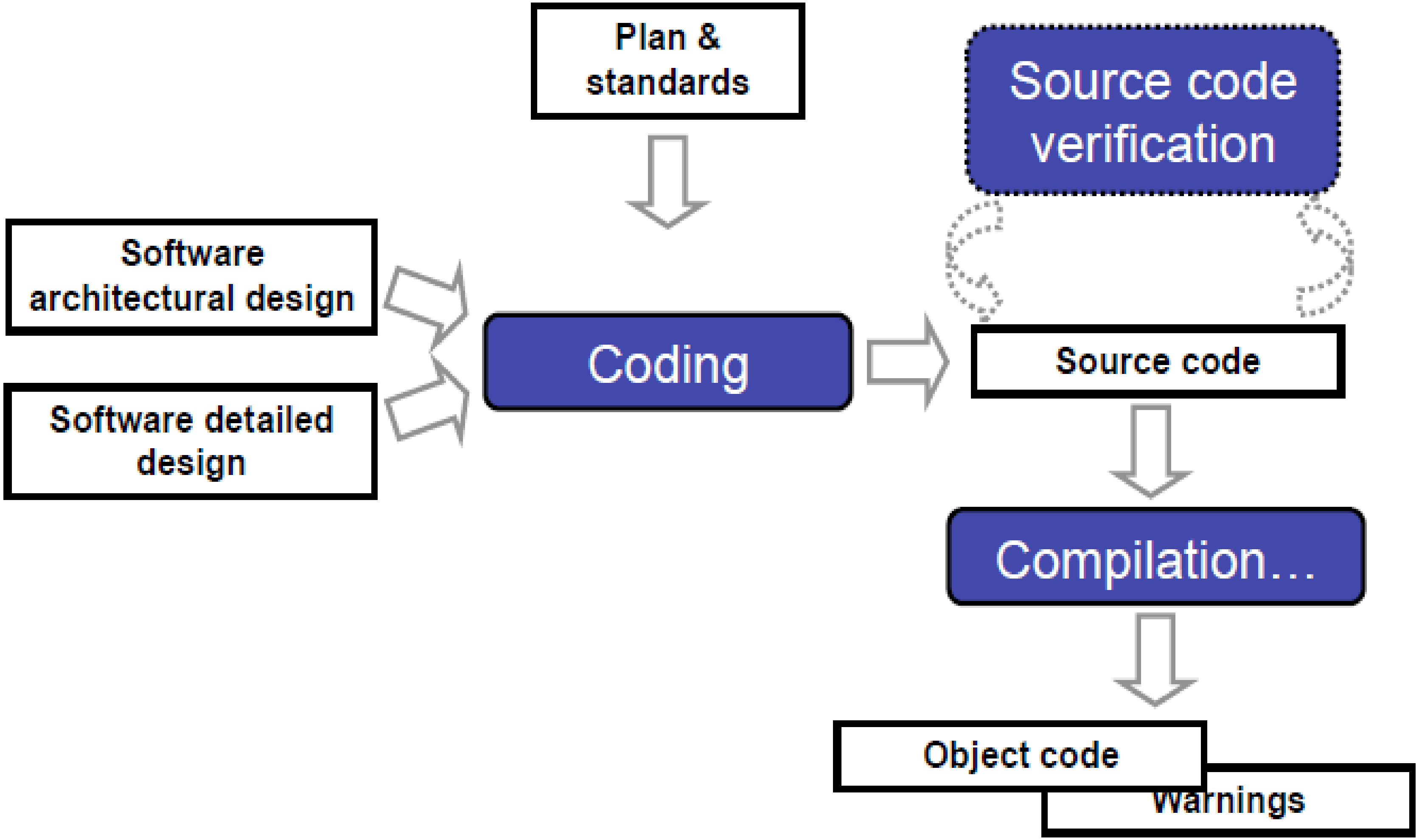


6 – Verification process – LLR Verification

The developer shall evaluate the software detailed design and test requirements considering:

- Traceability to the requirements of the software item
- External consistency with architectural design
- Internal consistency between software components and software units
- Non ambiguity, accuracy (type associated to data)
- Completeness
- Compliance to design standards
- Verifiability
- Compatibility with target computer
  - no conflicts between LLR 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
- Algorithm aspects

6 – Verification process – Coding Process Verification



Plan & standards

Software architectural design

Software detailed design

Coding

Source code verification

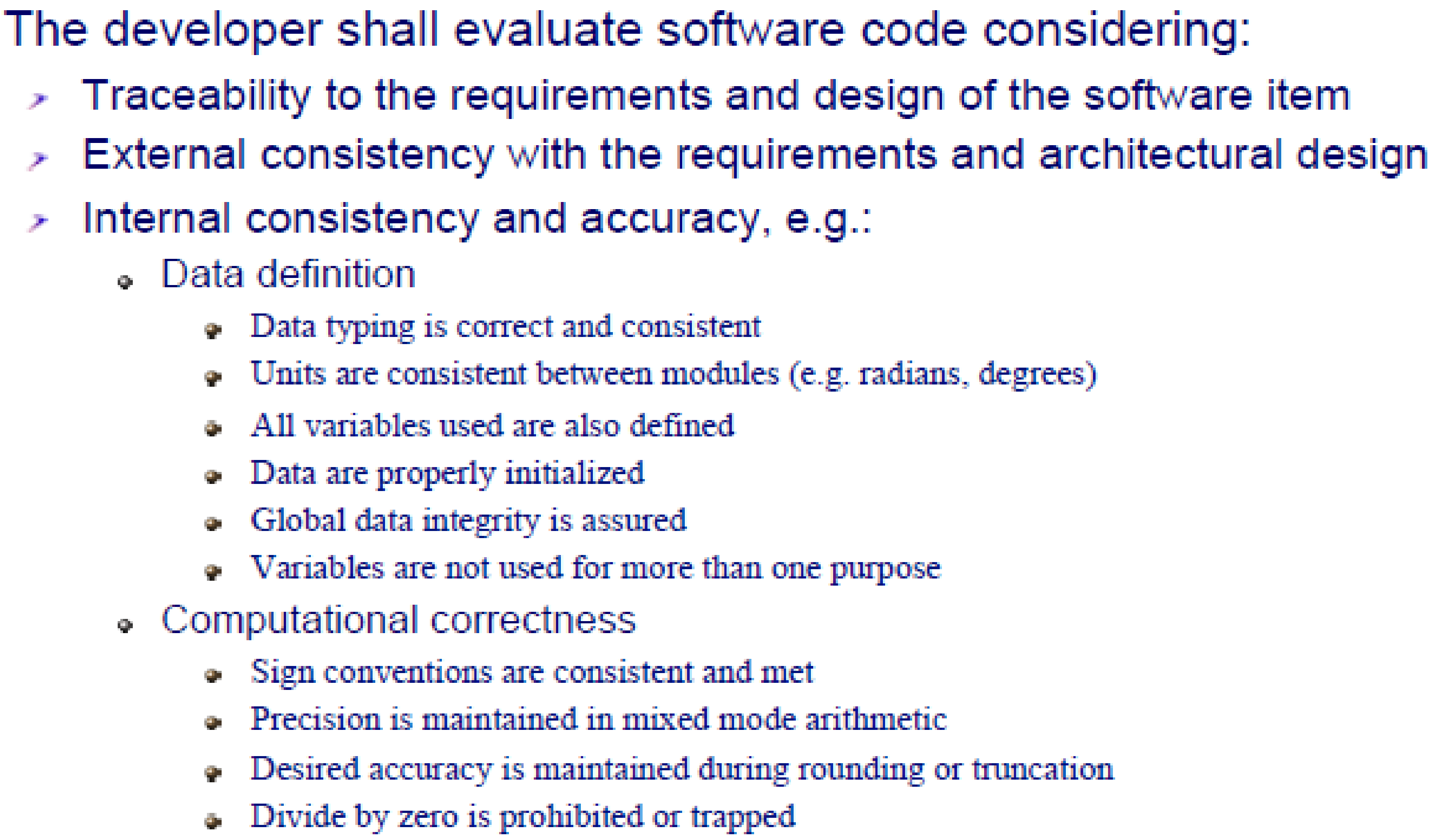
Source code

Compilation…

Object code

Warnings

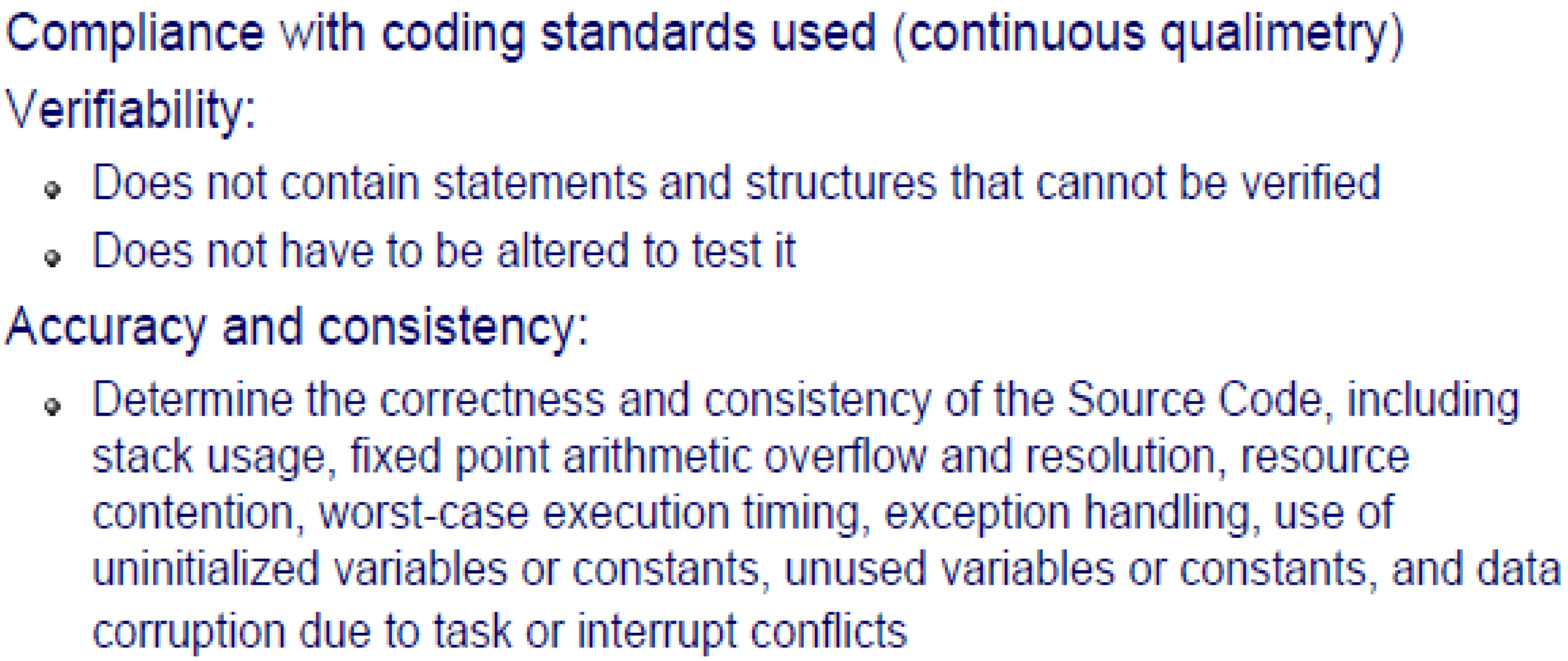
6 – Verification process – Source code Verification



The developer shall evaluate software code considering:

- Traceability to the requirements and design of the software item
- External consistency with the requirements and architectural design
- Internal consistency and accuracy, e.g.:
  - Data definition
    - Data typing is correct and consistent
    - Units are consistent between modules (e.g. radians, degrees)
    - All variables used are also defined
    - Data are properly initialized
    - Global data integrity is assured
    - Variables are not used for more than one purpose
  - Computational correctness
    - Sign conventions are consistent and met
    - Precision is maintained in mixed mode arithmetic
    - Desired accuracy is maintained during rounding or truncation
    - Divide by zero is prohibited or trapped

6 – Verification process – Source code Verification



Compliance with coding standards used (continuous qualimetry)

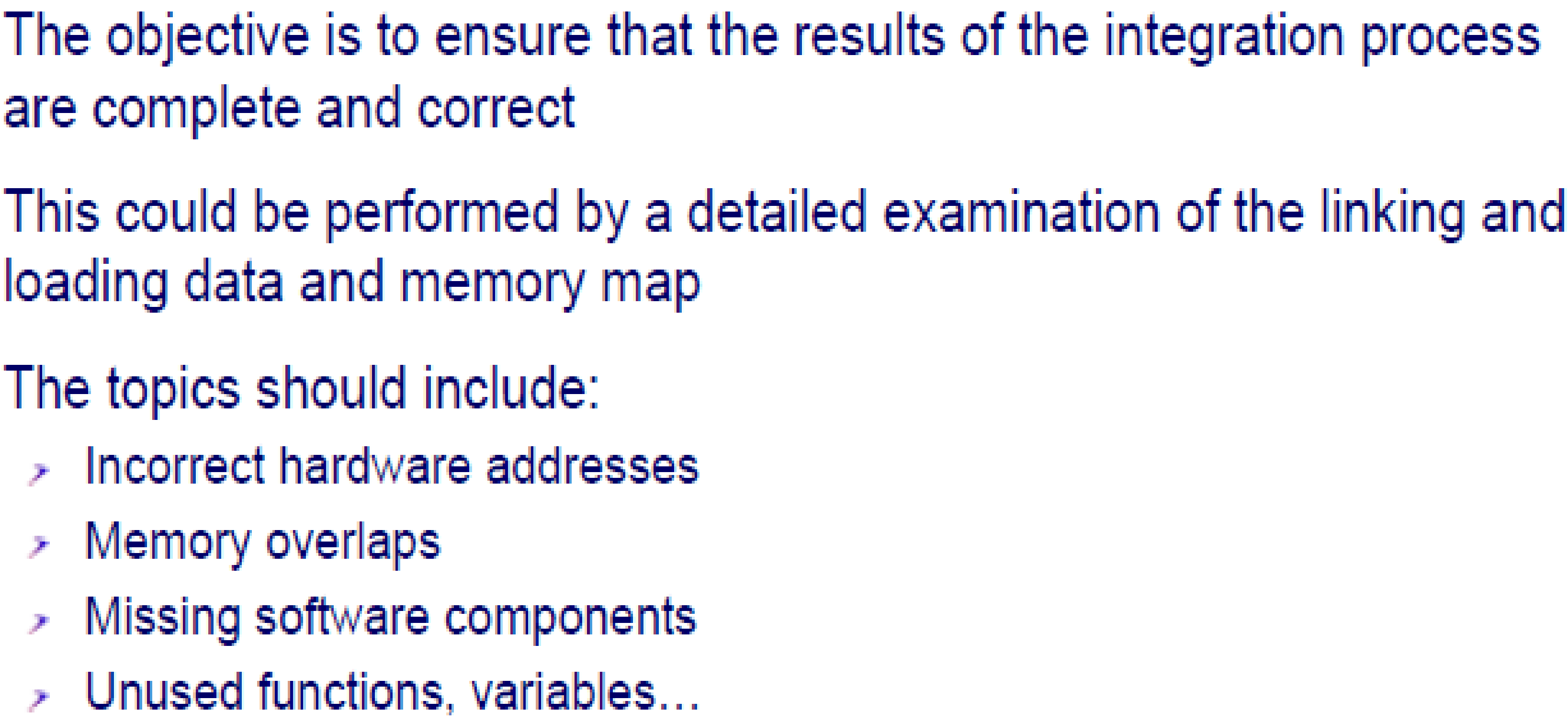
Verifiability:

- Does not contain statements and structures that cannot be verified
- Does not have to be altered to test it

Accuracy and consistency:

- 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 – Verification process – Integration Verification



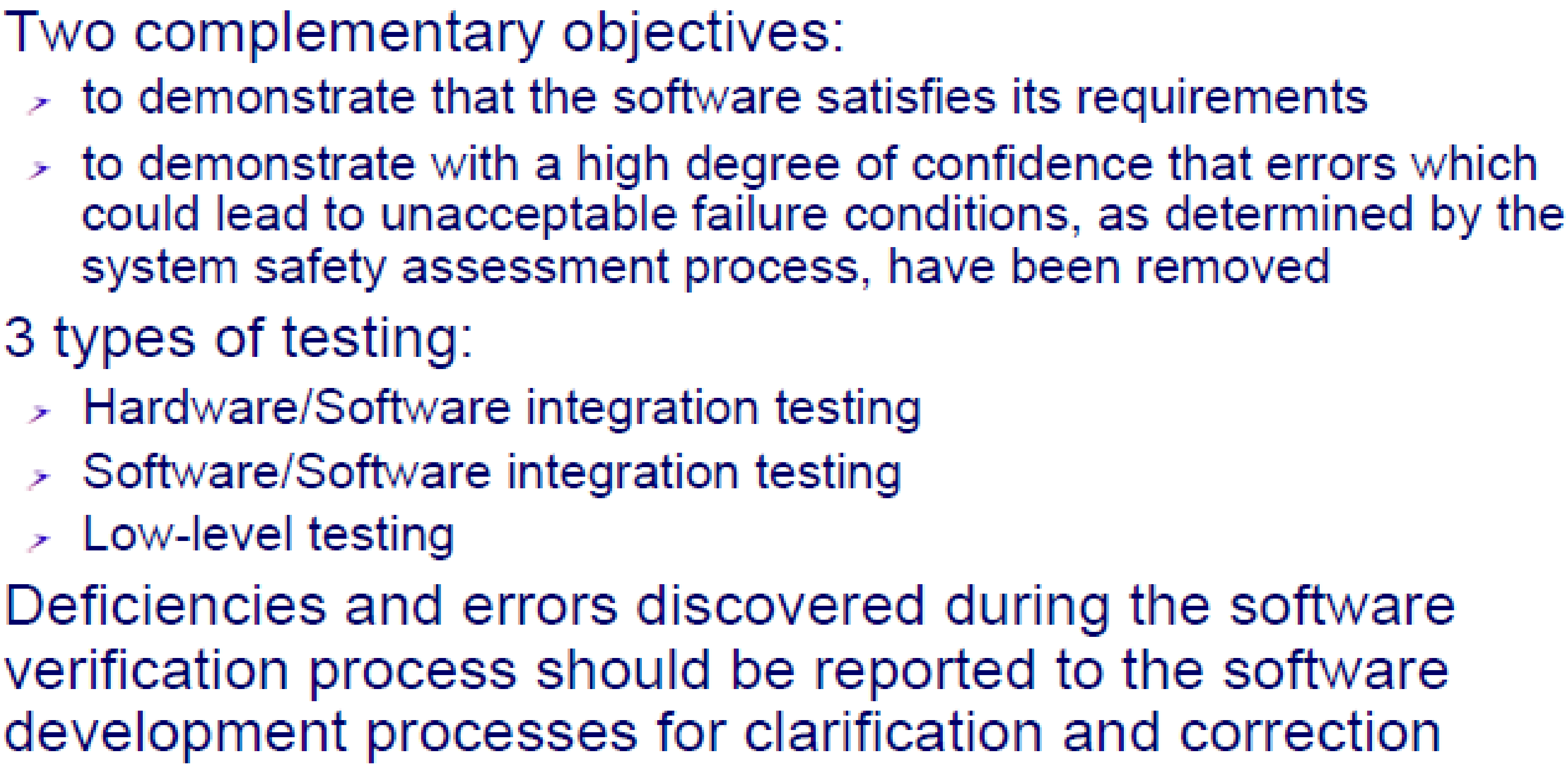
The objective is to ensure that the results of the integration process are complete and correct

This could be performed by a detailed examination of the linking and loading data and memory map

The topics should include:

- Incorrect hardware addresses
- Memory overlaps
- Missing software components
- Unused functions, variables…

6 – Verification process – Software Testing Procedure



Two complementary objectives:

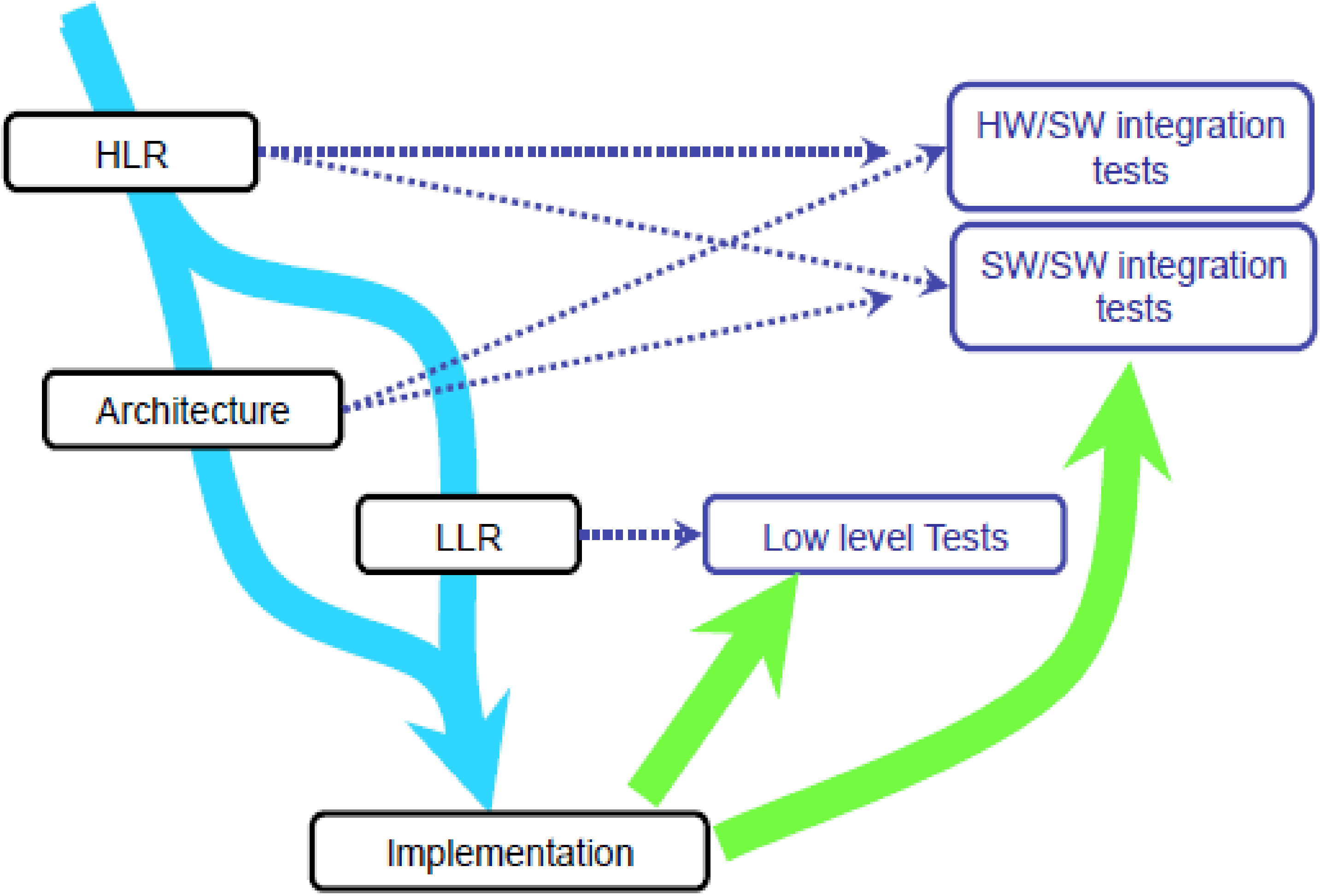
- to demonstrate that the software satisfies its requirements
- to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed

3 types of testing:

- Hardware/Software integration testing
- Software/Software integration testing
- Low-level testing

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

6 – Verification process – Software Testing Process



HLR

Architecture

LLR

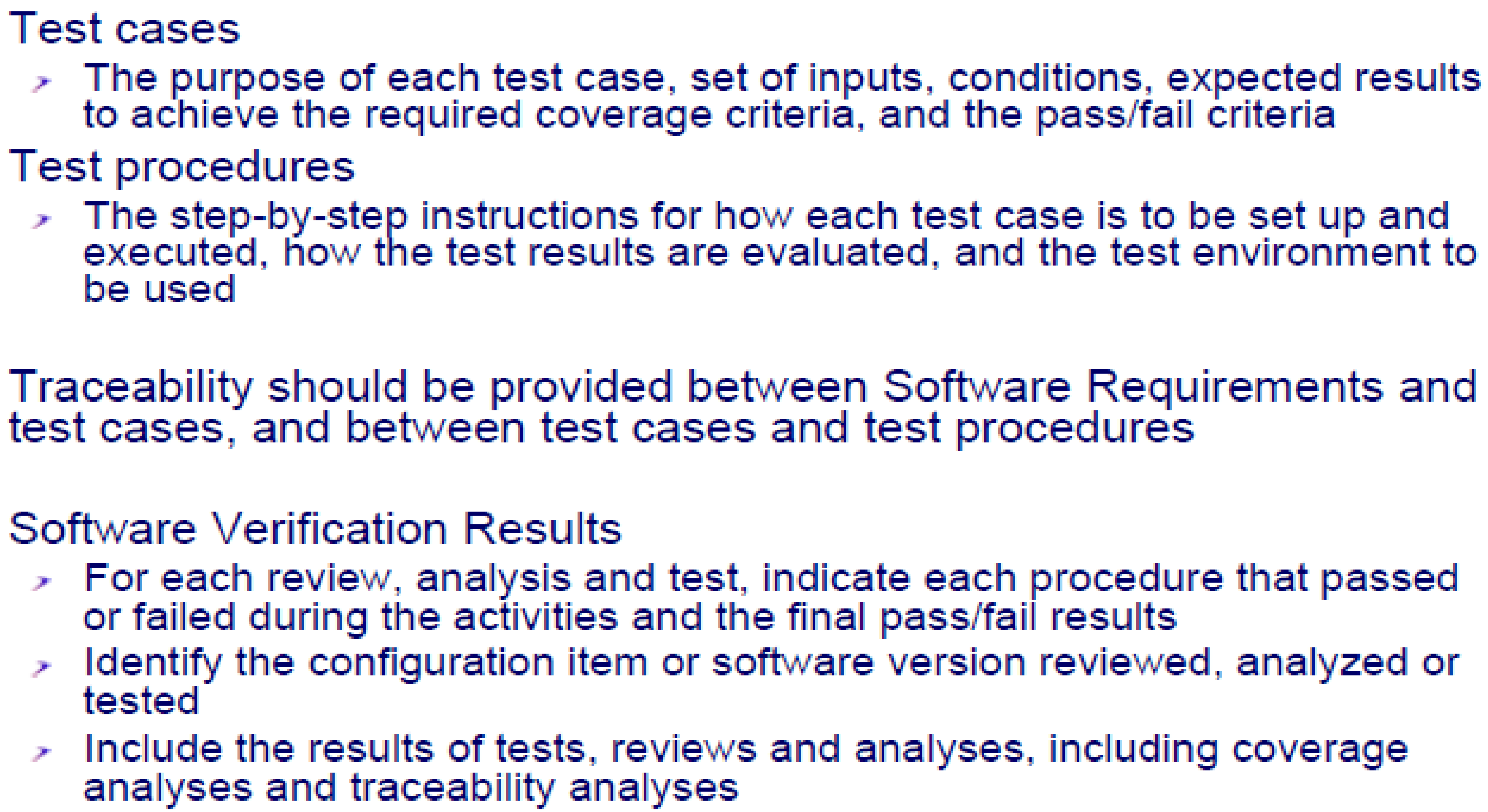
Implementation

HW/SW integration tests

SW/SW integration tests

Low level Tests

6 – Verification process – Software Testing Process



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

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

Traceability should be provided between Software Requirements and test cases, and between test cases and test procedures

Software Verification Results

- For each review, analysis and test, indicate each procedure that passed or failed during the activities and the final pass/fail results
- Identify the configuration item or software version reviewed, analyzed or tested
- Include the results of tests, reviews and analyses, including coverage analyses and traceability analyses

6 – Verification process

requirements based Testing process

Requirements-based testing known as the most effective at revealing errors

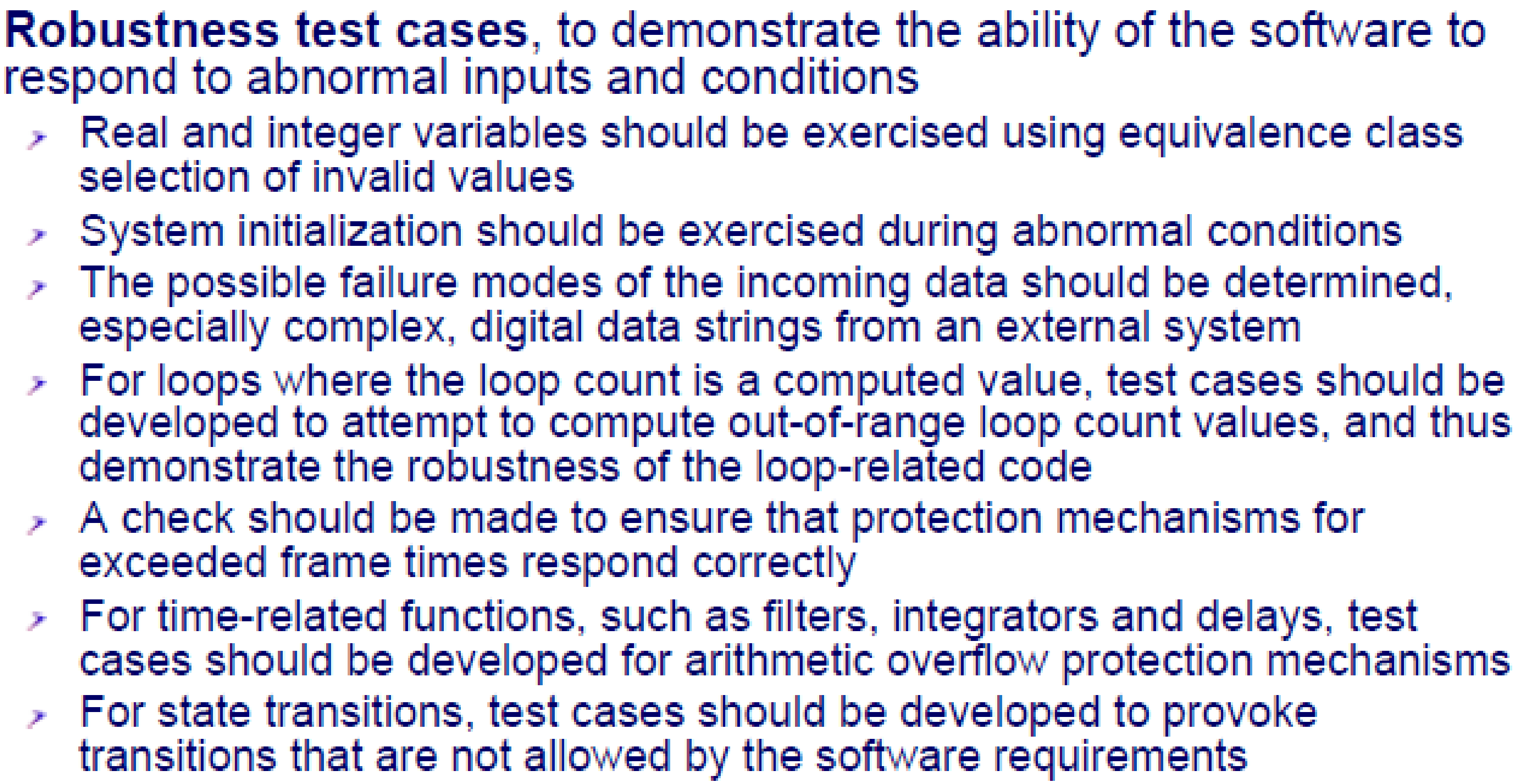
Two categories : Normal range and Robustness test cases

Normal range test cases, to demonstrate the ability of the software to respond to normal inputs and conditions

- Real and integer input variables should be exercised using valid ***equivalence classes*** and boundary values
- 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
- For state transitions, test cases should be developed to exercise the transitions possible during normal operation
- For software requirements expressed by logic equations, should verify the variable usage and the Boolean operators

6 – Verification process

requirements based Testing process



Robustness test cases, to demonstrate the ability of the software to respond to abnormal inputs and conditions

- Real and integer variables should be exercised using equivalence class selection of invalid values
- System initialization should be exercised during abnormal conditions
- The possible failure modes of the incoming data should be determined, especially complex, digital data strings from an external system
- 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
- A check should be made to ensure that protection mechanisms for exceeded frame times respond correctly
- For time-related functions, such as filters, integrators and delays, test cases should be developed for arithmetic overflow protection mechanisms
- For state transitions, test cases should be developed to provoke transitions that are not allowed by the software requirements

6 – Verification process – Structural coverage

To determine which code structure was not exercised by the requirements-based test procedures

Should confirm the degree of structural coverage appropriate to the software level

- Statement coverage
- Decision coverage
- Modified Condition/Decision Coverage (MCDC)

Should confirm the data coupling and control coupling between the code components

Level of Structural Coverage

- **« Statement coverage » / Software – level DAL C**
  - Each code statement is executed at least at least once
- **« Decision coverage » / Software – level DAL B**
  - Each program entry and exit point have been invoked at least once
  - Each decision takes all its values at least once
- **« Modified Conditions/Decisions coverage » / Software – level DAL A**
  - Each program entry and exit point have been invoked at least once
  - Each decision takes all its values at least once
  - Each condition in a decision takes all of its values at least once
  - Each condition independently affects the outcome of the decision
- **Condition :** A Boolean expression containing no Boolean operators except for the unary operator (NOT).
- **Decision :** A Boolean expression composed of conditions and zero or more Boolean operators. If a condition appears more than once in a decision, each occurrence is a distinct condition.

« Statement coverage »

```
If (((cond1 and cond2) or (cond3 and cond4)) and cond5) then
        result ;
End If
(no else statement)
```

statement coverage

cond1	cond2	cond3	cond4	cond5	Result
TRUE	**TRUE**	FALSE	TRUE	**TRUE**	**TRUE**

« Decision coverage »

```
If (((cond1 and cond2) or (cond3 and cond4)) and cond5) then
        result ;
End If
(no else statement)
```

Decision coverage

cond1	cond2	cond3	cond4	cond5	résultat
FALSE	TRUE	**FALSE**	TRUE	TRUE	**FALSE**
TRUE	**TRUE**	FALSE	**TRUE**	**TRUE**	**TRUE**

« Modified Conditions/Decisions coverage »

```
If (((cond1 and cond2) or (cond3 and cond4)) and cond5) then
        result ;
End If
```

(no else statement)

coverage "MC/DC"

cond1	cond2	cond3	cond4	cond5	result
FALSE	TRUE	FALSE	TRUE	TRUE	**FALSE**
TRUE	**TRUE**	FALSE	TRUE	TRUE	**TRUE**
TRUE	**FALSE**	**FALSE**	TRUE	TRUE	**FALSE**
TRUE	FALSE	**TRUE**	**TRUE**	**TRUE**	**TRUE**
TRUE	FALSE	TRUE	**FALSE**	TRUE	**FALSE**
TRUE	FALSE	TRUE	TRUE	**FALSE**	**FALSE**

Structural coverage result analysis

- **The structural coverage analysis can highlight:**
  - Incomplete test cases
    - ➢ Addition of test cases to complete the requirements coverage
  - Unreachable code structures
    - ➢ Justification to justify the presence of these code structures and verification of their coverage by analysis
  - The presence of dead code:
    - ➢ Dead code to be deleted, unless it is not present in the executable object code and if procedures are implemented to prevent its addition in the future construction of an executable

Dead code – Executable Object Code (or data) which exists as a result of a software development error but cannot be executed (code) or used (data) in any operational configuration of the target computer environment. It is not traceable to a system or software requirement.

Structural coverage result analysis

- **The structural coverage analysis can highlight:**
  - The presence of deactivated code for which:

    - The design has not planned its execution on aircraft or equipment

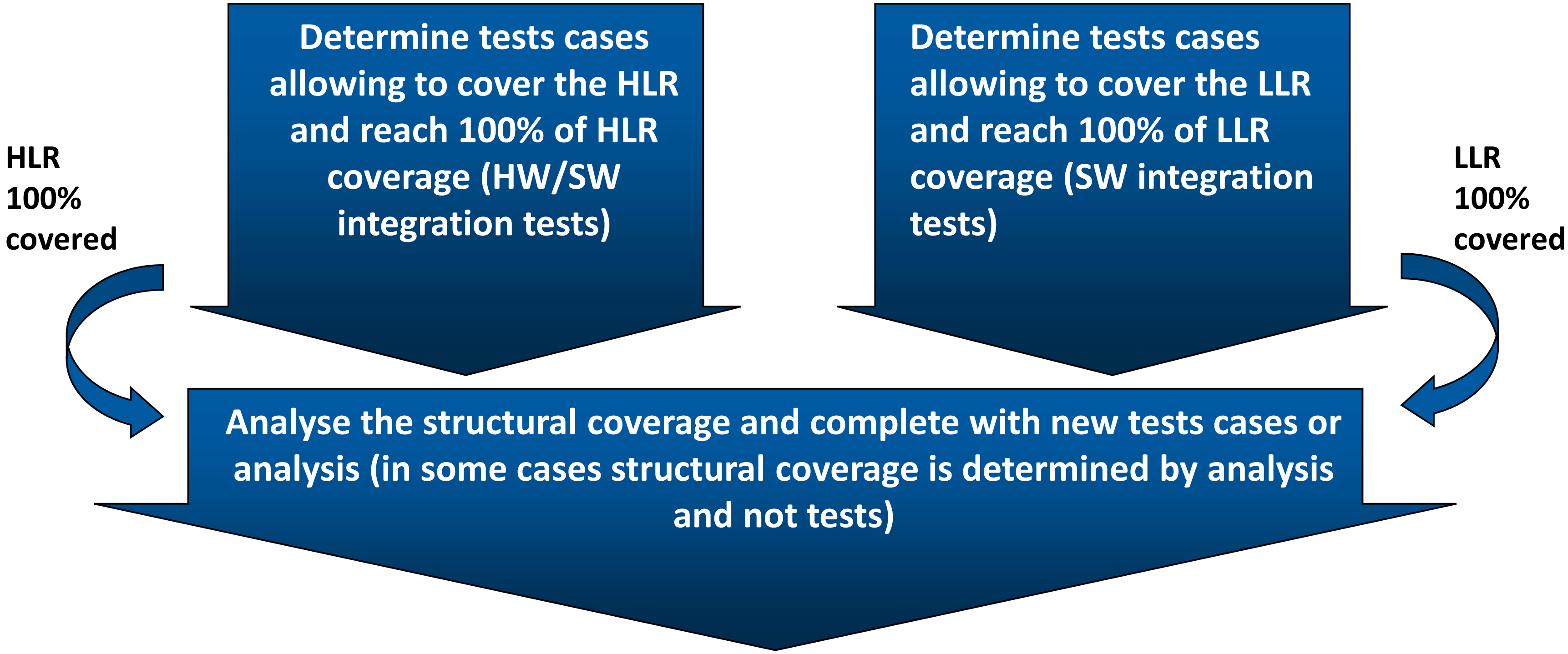
      ➢ Performing an analysis to ensure that inadvertent activation of the code is not possible

    - The design provides for execution in specific configurations of the aircraft or equipment

      ➢ Produce procedures and test cases to verify this disabled code

Deactivated code – Executable Object Code (or data) that is traceable to a requirement and, by design, is either (a) not intended to be executed (code) or used (data), for example, a part of a previously developed software component such as unused legacy code, unused library functions, or future growth code; 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.

Test strategy so-called « classic »



HLR 100% covered

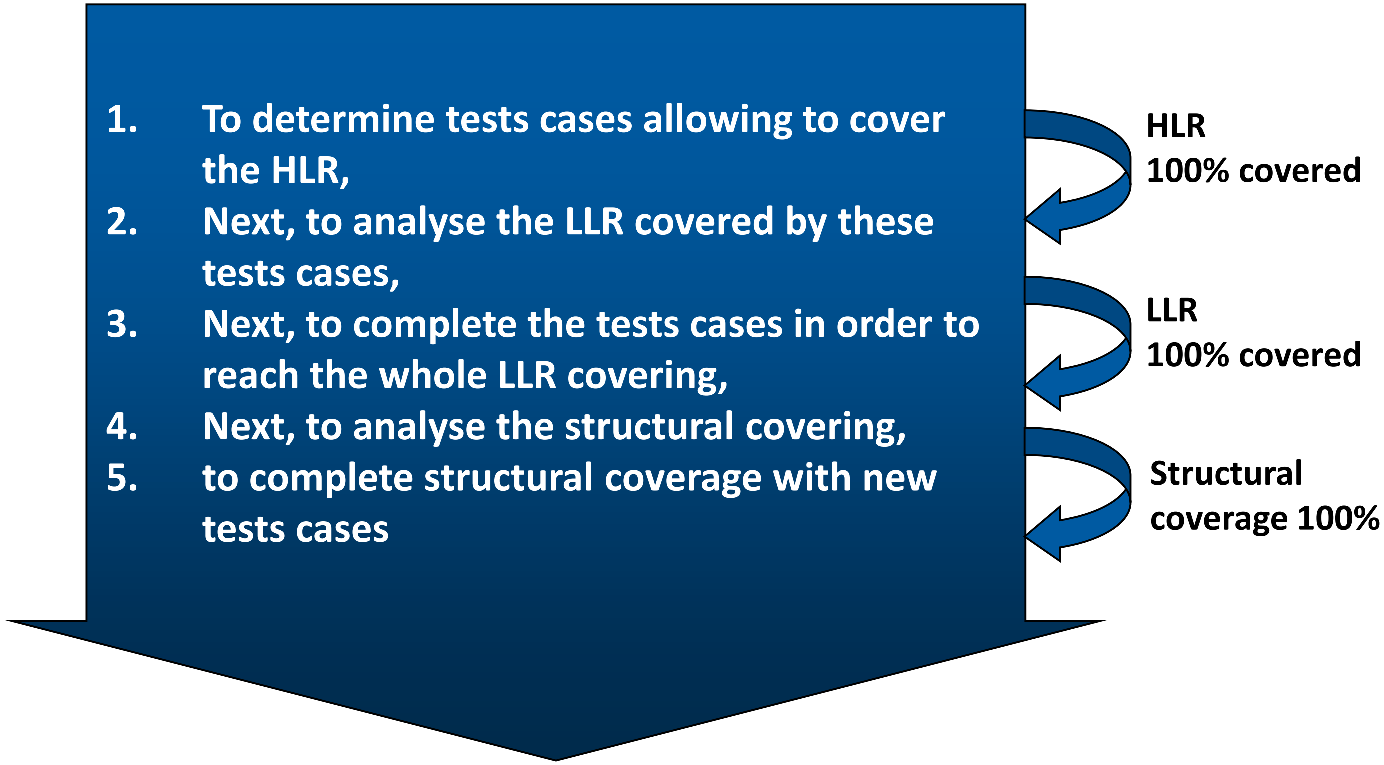
Determine tests cases allowing to cover the HLR and reach 100% of HLR coverage (HW/SW integration tests)

Determine tests cases allowing to cover the LLR and reach 100% of LLR coverage (SW integration tests)

LLR 100% covered

Analyse the structural coverage and complete with new tests cases or analysis (in some cases structural coverage is determined by analysis and not tests)

Test strategy so-called « Top Down »



1. To determine tests cases allowing to cover the HLR,
2. Next, to analyse the LLR covered by these tests cases,
3. Next, to complete the tests cases in order to reach the whole LLR covering,
4. Next, to analyse the structural covering,
5. to complete structural coverage with new tests cases

HLR
100% covered

LLR
100% covered

Structural
coverage 100%

Annexe A: Tables de synthèse - Objectif

- To provide objectives to be achieved during the development lifecycle and data that have to be produced. Objectives are specified in regard of the software level.

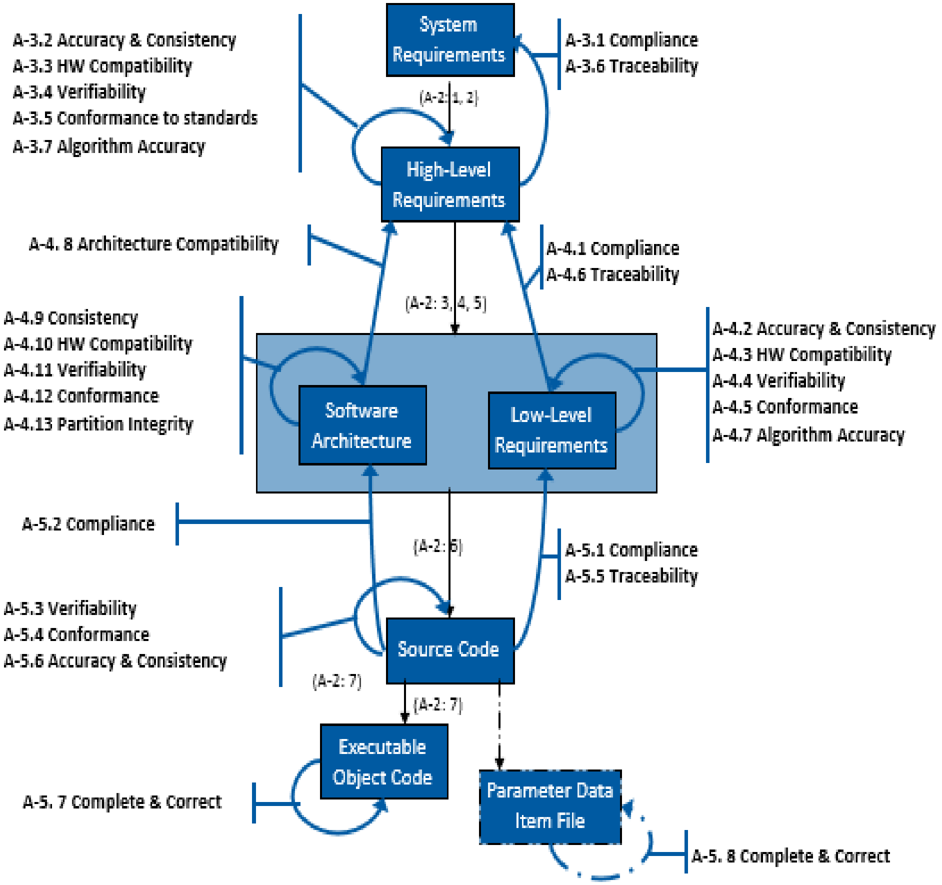
▪ 10 tables (A1 to A10) list the objectives and the output data of the lifecycle processes

▪ Tables includes recommendations for:

- Objectives applicable per process and per software level
- The independence level required per process and per software level
- The control category for the lifecycle data produced by each process.

Annexe A: Tables de synthèse

- Table A-1: Sofware planning process
- Table A-2: Software development
- Table A-3: Verification of Outputs of Software Requirements Process
- Table A-4: Verification of outputs of Software Design Process
- Table A-5: Verification of Outputs of Software Coding & Integration Processes
- Table A-6: Testing of Outputs of Integration Process
- Table A-7: Verification of Verification Process Results
- Table A-8: Software Configuration Management Process
- Table A-9: Software Quality Assurance Process
- Table A-10: Certification Liaison Process



System Requirements

A-3.2 Accuracy & Consistency
A-3.3 HW Compatibility
A-3.4 Verifiability
A-3.5 Conformance to standards
A-3.7 Algorithm Accuracy

A-3.1 Compliance
A-3.6 Traceability

(A-2: 1, 2)

High-Level Requirements

A-4. 8 Architecture Compatibility

A-4.1 Compliance
A-4.6 Traceability

(A-2: 3, 4, 5)

A-4.9 Consistency
A-4.10 HW Compatibility
A-4.11 Verifiability
A-4.12 Conformance
A-4.13 Partition Integrity

Software Architecture

Low-Level Requirements

A-4.2 Accuracy & Consistency
A-4.3 HW Compatibility
A-4.4 Verifiability
A-4.5 Conformance
A-4.7 Algorithm Accuracy

A-5.2 Compliance

(A-2: 6)

A-5.1 Compliance
A-5.5 Traceability

A-5.3 Verifiability
A-5.4 Conformance
A-5.6 Accuracy & Consistency

Source Code

(A-2: 7)

(A-2: 7)

Executable Object Code

Parameter Data Item File

A-5. 7 Complete & Correct

A-5. 8 Complete & Correct

How to read the summary tables?

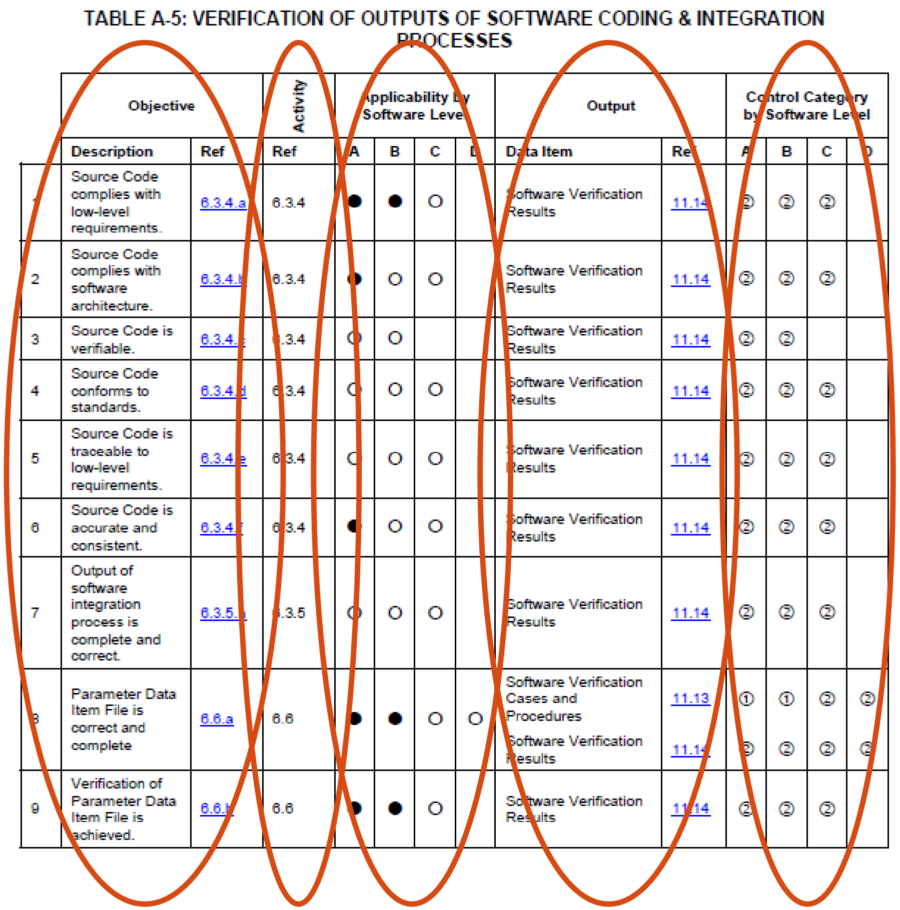


TABLE A-5: VERIFICATION OF OUTPUTS OF SOFTWARE CODING & INTEGRATION PROCESSES

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Source Code complies with low-level requirements.	6.3.4.a	6.3.4	●	●	○		Software Verification Results	11.14	②	②	②	
2	Source Code complies with software architecture.	6.3.4.b	6.3.4	●	○	○		Software Verification Results	11.14	②	②	②	
3	Source Code is verifiable.	6.3.4.c	6.3.4	○	○			Software Verification Results	11.14	②	②		
4	Source Code conforms to standards.	6.3.4.d	6.3.4	○	○	○		Software Verification Results	11.14	②	②	②	
5	Source Code is traceable to low-level requirements.	6.3.4.e	6.3.4	○	○	○		Software Verification Results	11.14	②	②	②	
6	Source Code is accurate and consistent.	6.3.4.f	6.3.4	●	○	○		Software Verification Results	11.14	②	②	②	
7	Output of software integration process is complete and correct.	6.3.5.a	6.3.5	○	○	○		Software Verification Results	11.14	②	②	②	
8	Parameter Data Item File is correct and complete	6.6.a	6.6	●	●	○	○	Software Verification Cases and Procedures Software Verification Results	11.13 11.14	① ②	① ②	② ②	② ②
9	Verification of Parameter Data Item File is achieved.	6.6.b	6.6	●	●	○		Software Verification Results	11.14	②	②	②	

Reminder for the process, of all the objectives to be reached with reference to the dedicated section of the document

Reference to § for the description of the activity

Reminder of the data to be produced and reference to §

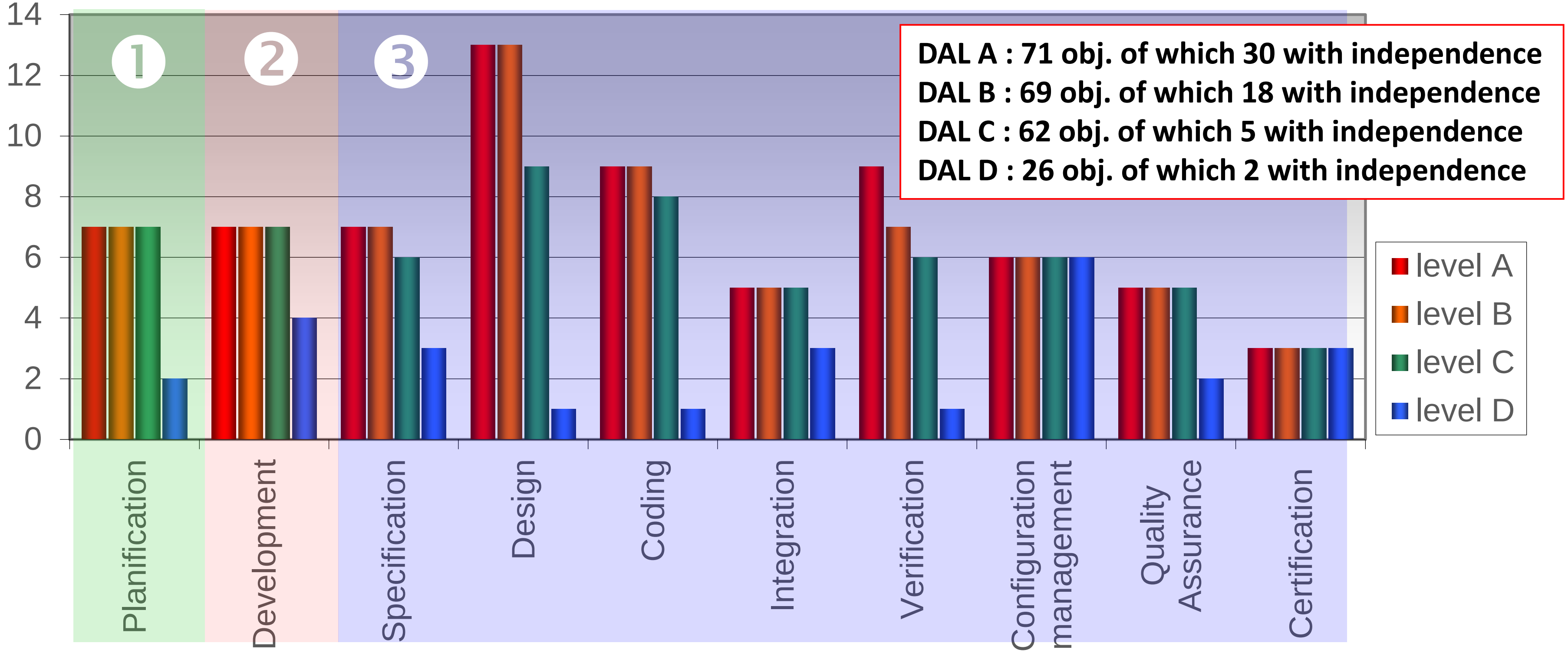
Configuration rules (1) or (2) according to DAL

Applicability according to the DAL

- ○ Required activity
- ● Required activity with independence

Objectives vs DAL

- Number of objectives to be reached per phase and per DAL



DAL A : 71 obj. of which 30 with independence
DAL B : 69 obj. of which 18 with independence
DAL C : 62 obj. of which 5 with independence
DAL D : 26 obj. of which 2 with independence

- No significant differences between a DAL A level and a DAL B level on the number of objectives to be achieved. The main difference concerns the degree of independence with which these objectives shall be covered.

What independence between activities?

- « <u>Independence</u> : 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. »

INDEPENDENCE

Writing Requirements — Validation & verification (analyzes, reviews, Test execution)

Coding — Analyzes, code reviews

Writing and test implementation — Analyzes, test review

Quality Assurance Process — Other Process (management, development, verification, configuration)

- More the verifier of an item is independent from the one who developed this item, less the verifier can make the same mistakes, because of:
  - Interpretation due to the use of vocabulary,
  - A assertion due to the knowledge of the context.

11 –Life cycle development - Plan and standard

Plans : what, when, who and how

Standards : rules

PLANS

- **Plan for Software Aspects of Certification (PSAC)**
- **Software Development Plan (SDP)**
  Describes the software development strategy
- **Software Verification Plan (SVP)**
  Describes the software verification strategy
- **Software Configuration Management Plan (SCMP)**
  Describes the provisions for software Configuration Management
- **Software Quality Assurance Plan (SQAP)**
  Describes the software Quality Assurance provisions

STANDARDS

- **Software Requirements Standards (SRS)**
  Defines the rules for software specification
- **Software Design Standards (SDS)**
  Defines the rules for software conception
- **Software Code Standards (rules) (SCS)**
  Defines the rules for software coding

Safeguards against the implementation of risky development practices

Plan for Software Aspect of Certification (PSAC)

- **Input document for certification authorities:**
  - It summarizes the software development processes as described in the other plans
  - It gives a global view of the system and software architecture It presents the considerations relating to certification (level of DAL, applicable standard)
  - It details the software development cycle and its data
  - It specifies additional considerations: use of COTS or software already developed previously, qualification of the tools It indicates the project schedule
  - It describes the activities carried out by the subcontractors

Software Development Plan (SDP)

- **Input document to describe activities related to software development (dedicated to development teams):**
  - It defines the detailed organization of the project, the teams and the associated responsibilities for the development activities
  - It identifies the standards to be followed (SRS, SDS, SCS, others…)
  - It describes the software development cycle, the development phases (Specification, Design, Coding, Integration) by including the transition criteria between these phases
  - It describes the environment of the Software development cycle: development tools, compilers, linkers, GAC, verification tools, conf management tools, traceability tools, etc.
  - It provides a description of the H / W platform

Software Verification Plan (SVP)

- **Input document to describe the activities related to software verification (dedicated to dev and verification teams):**
  - It defines the detailed organization of the project, the teams and the associated responsibilities for verification activities
  - It describes the verification activities (analysis, reviews, tests), including the transition criteria
  - It describes the verification environment (testing tools, analysis tools, etc.)
  - It identifies considerations on the independence of Verification activities
  - It describes procedures to define the verification activities when the software is modified

Software Configuration Management Plan (SCMP)

- **Input document to describe the provisions relating to software configuration management (dedicated to dev. And verification teams):**
  - It describes the configuration management environment (procedures, tools, methods, responsibilities and interfaces with other processes)
  - It describes the activities relating to configuration management (configuration identification, management of Baselines and traceability, modification management, problem management, data archiving and restoration, loading of software versions, etc.)
  - It defines the transition criteria to initiate the Configuration Management process.

Software Quality Assurance Plan (SQAP)

- **Input document to describe the provisions relating to quality assurance (dedicated to quality teams):**
  - It defines the methods and tools to ensure:
    - ➢ software development meets DO-178C, plans and standards
    - ➢ compliance of software deliveries
  - It describes the quality activities by phase
  - It identifies the Quality Assurance contact within the Development processes
  - It specifies how quality records are managed
  - It describes the means implemented to verify the correct application of the processes by the subcontractors

Software Requirement Standards (SRS)

- **Input document for the development of specification requirements**
  - It describes the formalism to be used to express requirements, such as given flow diagrams, formal specification languages
  - It specifies the constraints of using the specification tools

Software Design Standards (SDS)

- **Input document for the development of design requirements and the definition of software architecture**
  - It describes the rules, methods and tools to be applied to develop the software design (eg UML notation, HOOD; use of the IT mechanism, exceptions; ...)
  - It describes the restrictions imposed on authorized design methods (e.g. scheduling, architecture driven by interruptions or events, dynamic allocation of tasks, reentrancy, use of global data, exception mechanism) and the justification
  - It presents the design constraints (prohibition of recursion, dynamic objects)
  - It specifies the constraints of using design tools

Software Code Standards (SCS)

- **Input document for software coding**
  - It specifies the programming language used and the limitations regarding certain risky uses of the language
  - It describes the rules for writing source code (line size, indentation rules, revision history, etc.)
  - It specifies the naming conventions for components, subroutines, variables, constants, etc.
  - It specifies the requirements in terms of performance
  - It describes the constraints to be respected, such as the degree of coupling between the components, the cyclomatic complexity and the justification for their use.
  - It specifies the constraints of using coding tools

Objectives to achieve

- Plans are required for all levels

- Standards are not required for level D
- The plan review is reduced for level D

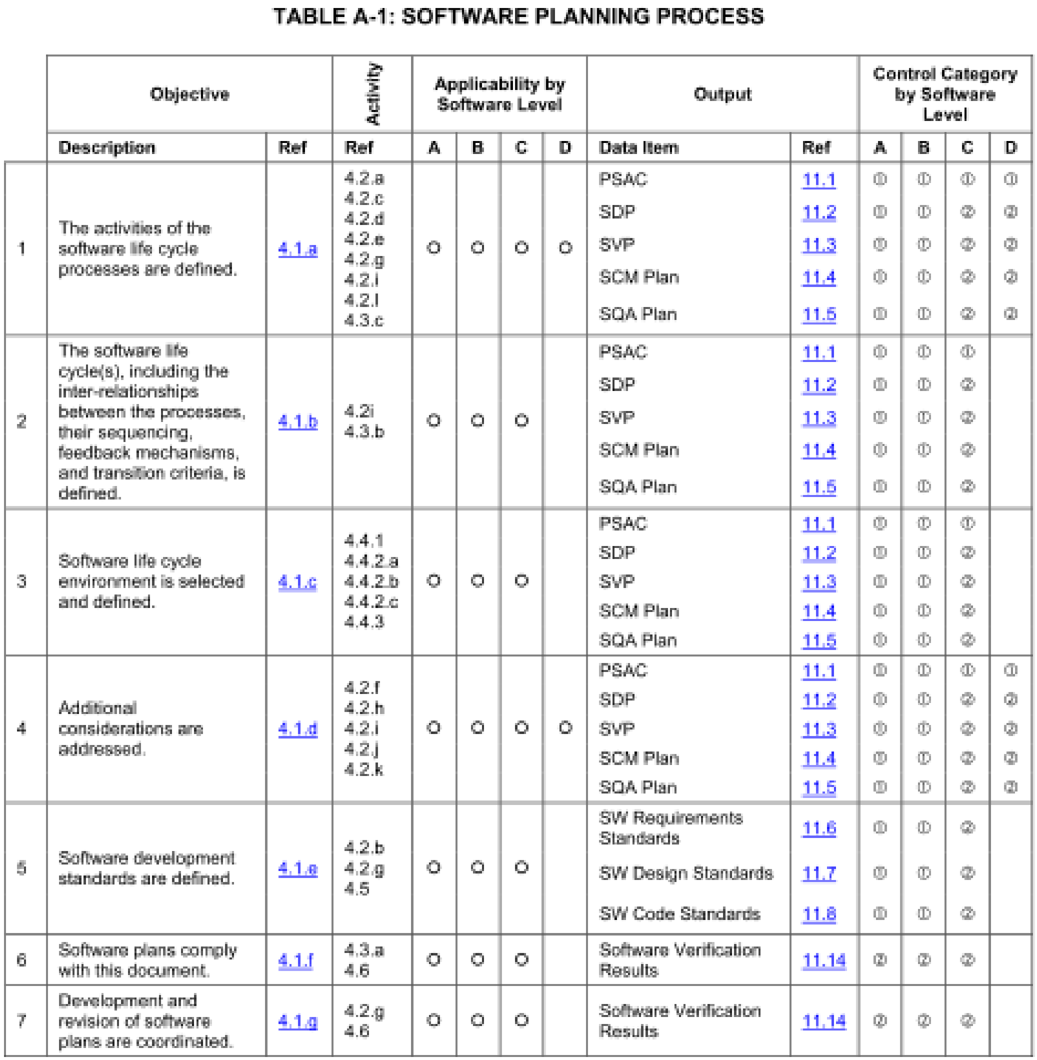


TABLE A-1: SOFTWARE PLANNING PROCESS

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	The activities of the software life cycle processes are defined.	4.1.a	4.2.a 4.2.c 4.2.d 4.2.e 4.2.g 4.2.i 4.2.l 4.3.c	○	○	○	○	PSAC	11.1	①	①	①	①
								SDP	11.2	①	①	②	②
								SVP	11.3	①	①	②	②
								SCM Plan	11.4	①	①	②	②
								SQA Plan	11.5	①	①	②	②
2	The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, is defined.	4.1.b	4.2i 4.3.b	○	○	○		PSAC	11.1	①	①	①	
								SDP	11.2	①	①	②	
								SVP	11.3	①	①	②	
								SCM Plan	11.4	①	①	②	
								SQA Plan	11.5	①	①	②	
3	Software life cycle environment is selected and defined.	4.1.c	4.4.1 4.4.2.a 4.4.2.b 4.4.2.c 4.4.3	○	○	○		PSAC	11.1	①	①	①	
								SDP	11.2	①	①	②	
								SVP	11.3	①	①	②	
								SCM Plan	11.4	①	①	②	
								SQA Plan	11.5	①	①	②	
4	Additional considerations are addressed.	4.1.d	4.2.f 4.2.h 4.2.i 4.2.j 4.2.k	○	○	○	○	PSAC	11.1	①	①	①	①
								SDP	11.2	①	①	②	②
								SVP	11.3	①	①	②	②
								SCM Plan	11.4	①	①	②	②
								SQA Plan	11.5	①	①	②	②
5	Software development standards are defined.	4.1.e	4.2.b 4.2.g 4.5	○	○	○		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.	4.1.f	4.3.a 4.6	○	○	○		Software Verification Results	11.14	②	②	②	
7	Development and revision of software plans are coordinated.	4.1.g	4.2.g 4.6	○	○	○		Software Verification Results	11.14	②	②	②	

Sommaire

- The evolution of the aeronautical world
- Presentation of document
- Relation with other guides and level of criticality
- DO 178 Process
- **Liaison with certification authorities**

Liaison with certification authorities

- Certification authorities :
  - Federal Aviation Administration (FAA) for USA
  - European Aviation Safety Agency (EASA) for European Union
- Certification Objectives:
  - Ensure a high and consistent level of safety for civil flight
  - Co-operate with authorities of all nations to ensure safe flights around the world.
- Software certification :
  - The software is considered as part of an aeronautical equipment or system and not as a single independent product
  - The certification of a software is for a given type of aircraft or machine and shall be reconsidered for each type of aircraft or machine in which the software is embedded

Liaison with certification authorities

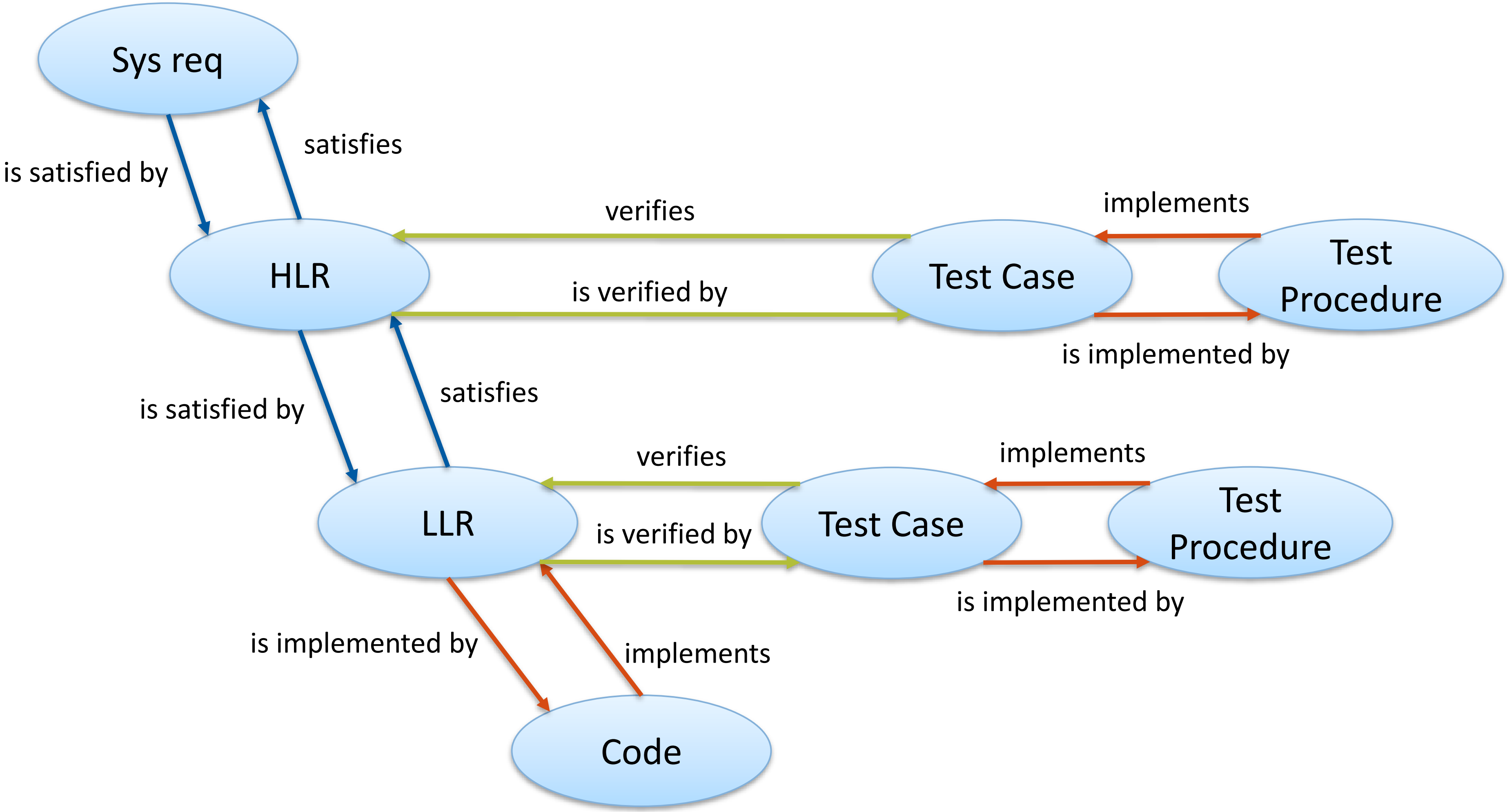
- Certification Liaison Process :
  - Ensures communication between the applicant and the certification authority
- Role of implementer :
  - Uses DO-178 as a set of recommendations to meet certification objectives
  - Submit software life cycle data to the certification authority
  - Identifies the processes, activities, methods and tools to be implemented
  - Ensures the performance of these processes

Liaison with certification authorities

- Role certification authority
  - Gives acceptance of criticality level (DAL: design assurance level) of software for equipment or system
  - Validates software compliance with DO178 requirements based on the provisions described in the project plans.

Annexes

- ## Data model & traceability links


Sys req
is satisfied by
satisfies
HLR
verifies
is verified by
Test Case
implements
Test Procedure
is implemented by
is satisfied by
satisfies
LLR
verifies
is verified by
Test Case
implements
Test Procedure
is implemented by
is implemented by
implements
Code