Software architecture in ASPICE and 26262

Even-André Karlsson

addalot
QUALITY IMPROVEMENT

Agenda

- Overall comparison (3 min)
- Why is the architecture documentation difficult? (2 min)
- ASPICE requirements (8 min)
- 26262 requirements (12 min)
- Comparison and questions (5 min)
- Discussion tomorrow in workshop

addalot QUALITY IMPROVEMENT

Relation between ASPICE and 26262

- ASPICE is a maturity capability standard with a large coverage
  - No generic practices in 26262, but some level 2 & 3 requirements on tailoring and selection
- 26262 is a safety standard with increasing requirements depending on ASIL level
- 26262: Little focus on organizational processes, metrics or improvements – one project and product at the time – but having an ASPICE maturity helps
- 26262: Much more focus on technical practices to ensure safety – ASPICE only talks about generic requirements
- Requirements on traceability are stronger in ASPICE than in 26262
- ASPICE is a small standard (SW arch: less than 2 + 0,5 = 2,5 pages), 26262 is large (SW arch: 6 + 2 (App D) + 10 (part 9) = 18 pages)
- Much overlap, e.g. architecture

Why is documenting the SW arch difficult?

- SW developers are not used to documenting the architecture
- How to combine with agile development?
- How to incorporate all Safety documentation?
- Maintaining the documentation
- Distinguish between System and Software level
- Level of detail

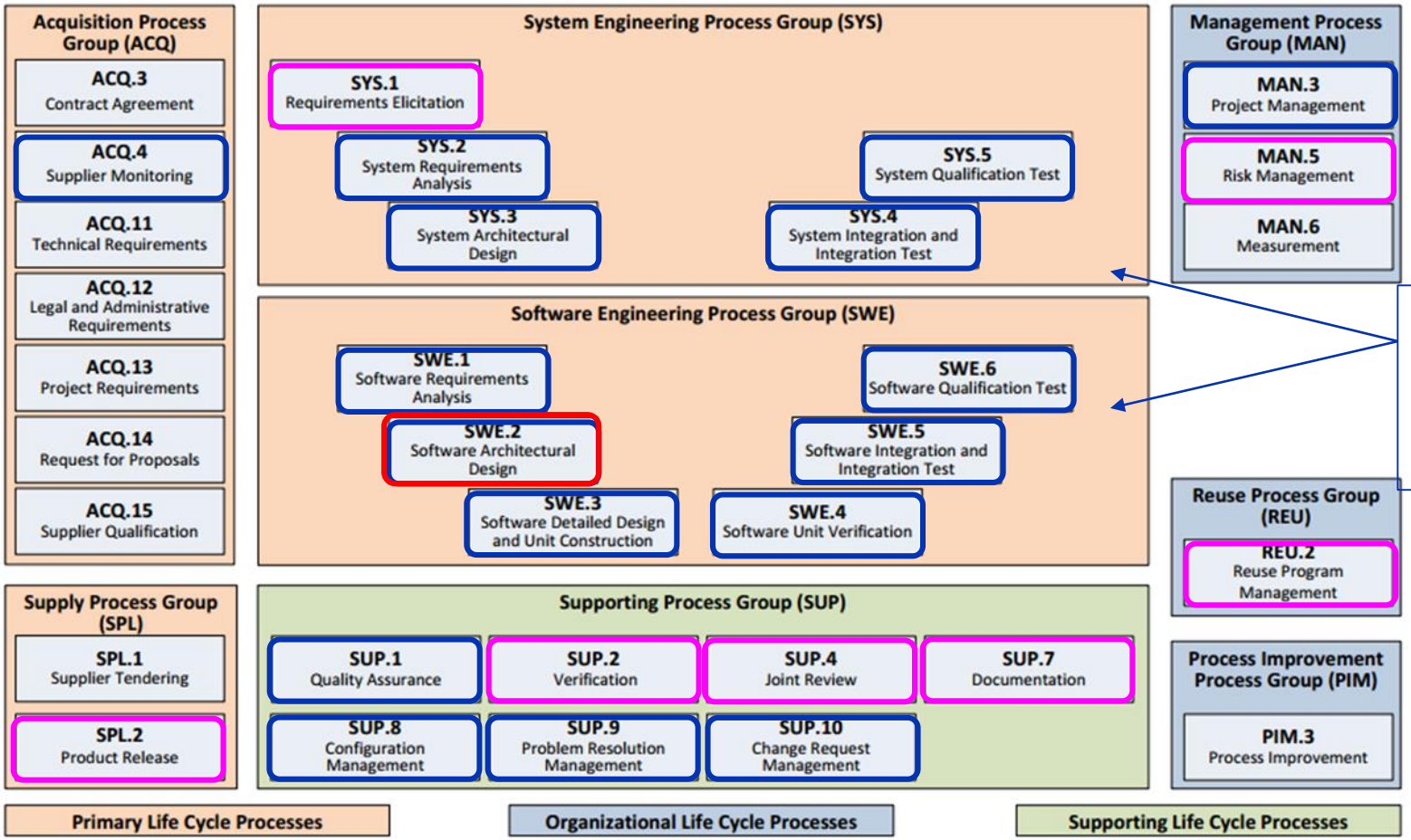
Further to be discussed in workshop tomorrow!

This only covers SW architecture – SYSTEM is actually more interesting

Automotive SPICE (Version 3.0)

HIS Scope

Extended HIS Scope

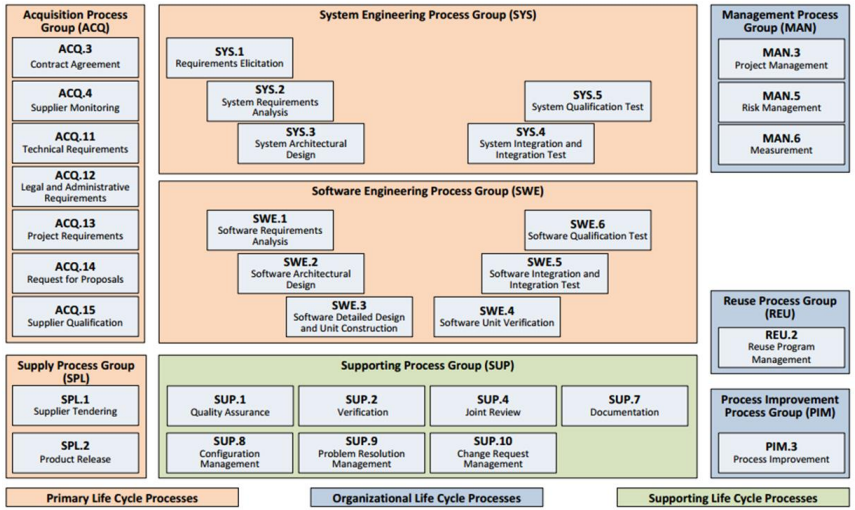


Previously Eng 1-10 Changes in Test structure

addalot QUALITY IMPROVEMENT

Related ASPICE processes

- System and Software Requirements
- System architecture
- Test strategy (from all test processes)
- Verification = Review
- Traceability



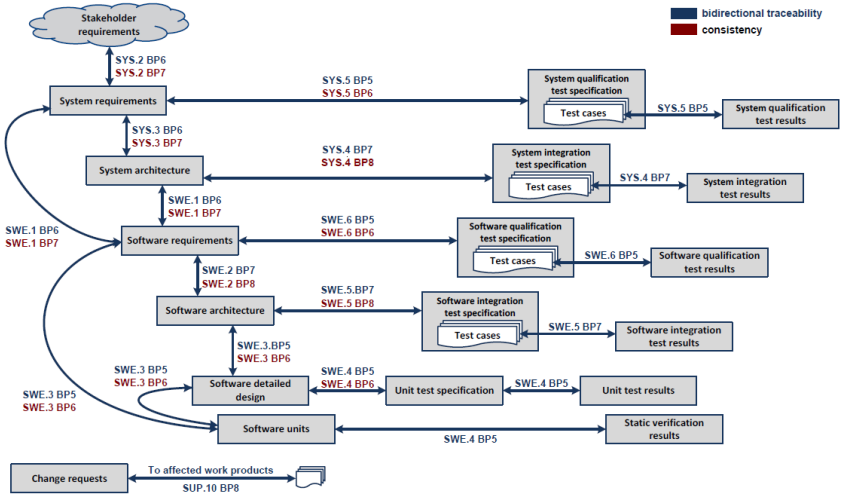


Figure D.4 — Bidirectional Traceability and Consistency

QUALITY IMPROVEMENT

SW architecture work product

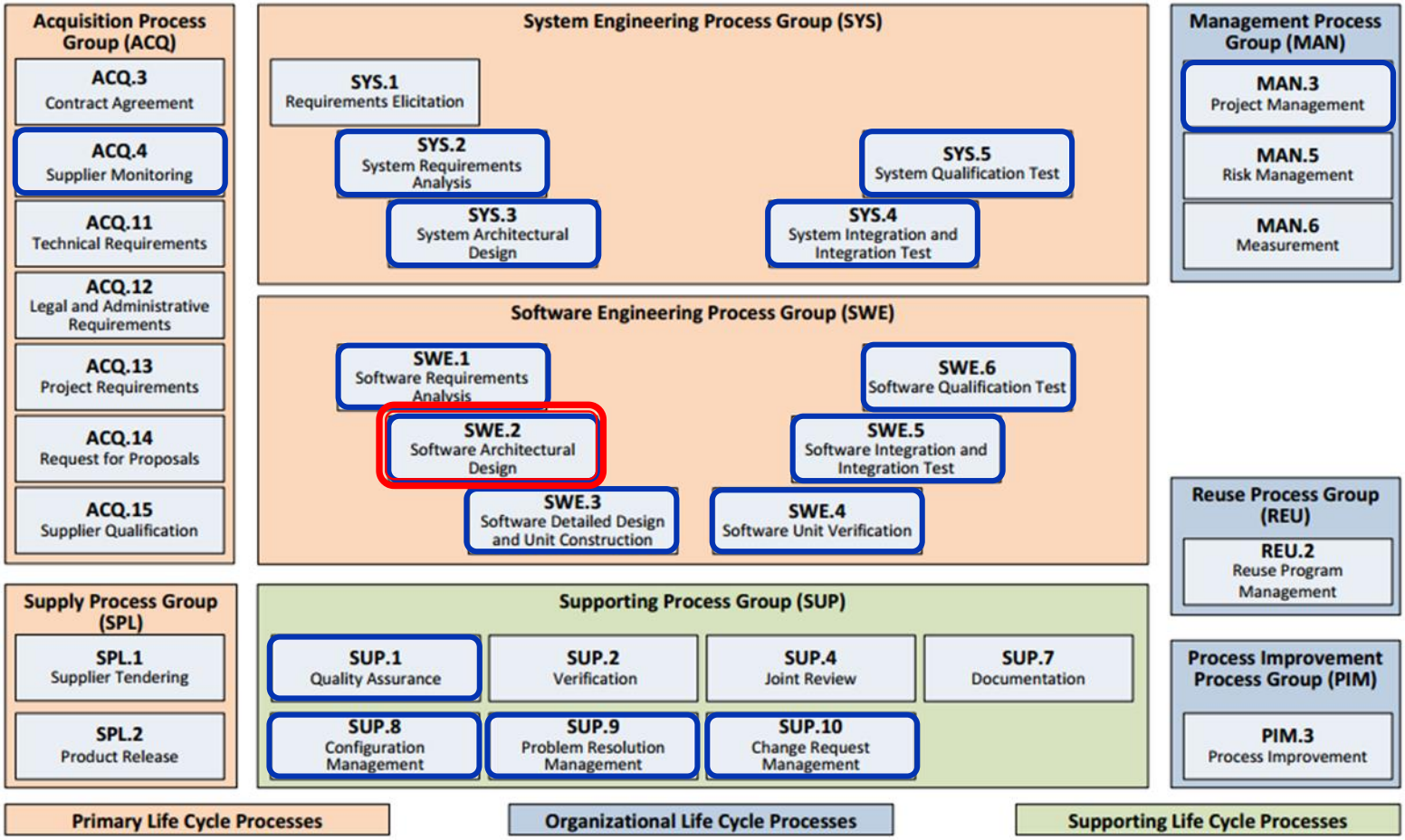
04-03	Domain model	• Must provide a clear explanation and description, on usage and properties, for reuse purposes • Identification of the management and structures used in the model
04-04	Software architectural design	• Describes the overall software structure • Describes the operative system including task structure • Identifies inter-task/inter-process communication • Identifies the required software elements • Identifies own developed and supplied code • Identifies the relationship and dependency between software elements • Identifies where the data (such as parameters) are stored and which measures (e.g. checksums, redundancy) are taken to prevent data corruption • Describes how variants for different model series or configurations are derived • Describes the dynamic behavior of the software (Start-up, shutdown, software update, error handling and recovery, etc.) • Identifies where the data (such as parameters) are stored and which measures (e.g. checksums, redundancy) are taken to prevent data corruption • Describes which data is persistent and under which conditions • Consideration is given to: - any required software performance characteristics - any required software interfaces - any required security characteristics required - any database design requirements
04-05	Software detailed design	• Provides detailed design (could be represented as a prototype, flow chart, entity relationship diagram, pseudo code, etc.) • Provides format of input/output data

Note duplication ☺
Same in 2.5 and 3.0

addalot QUALITY IMPROVEMENT

Automotive SPICE Reference Model 3.0

SWE.2 – Software Architectural Design



addalot QUALITY IMPROVEMENT

SWE.2 – Software Architectural Design

Process Purpose: The purpose of the Software Architectural Design Process is to establish an architectural design and to identify which software requirements are to be allocated to which elements of the software, and to evaluate the software architectural design against defined criteria.

Typical Challenges:

1. What level of details is needed
2. Find a healthy level of modules - identifying all necessary modules and interfaces
3. Creating an architecture that is flexible and adoptable to change
4. Finding balance between flexibility and performance – lots of layers and indirection will makes things slower (important for embedded systems)
5. Evaluation criteria and recording design decisions
6. Traceability
7. Keeping the architecture documentation up-to-date

addalot QUALITY IMPROVEMENT

SWE.2 – Software Architectural Design

Base Practices:

- **SWE.2.BP1: Develop software architectural design.** Develop and document the software architectural design that specifies the elements of the software with respect to functional and non-functional software requirements.
- **SWE.2.BP2: Allocate software requirements.** Allocate the software requirements to the elements of the software architectural design.
- **SWE.2.BP3: Define interfaces of software elements.** Identify, develop and document the interfaces of each software element.
- **SWE.2.BP4: Describe dynamic behavior.** Evaluate and document the timing and dynamic interaction of software elements to meet the required dynamic behavior of the system.
- **SWE.2.BP5: Define resource consumption objectives.** Determine and document the resource consumption objectives for all relevant elements of the software architectural design on the appropriate hierarchical level.
- **SWE.2.BP6: Evaluate alternative software architectures.** Define evaluation criteria for architecture design. Evaluate alternative software architectures according to the defined criteria. Record the rationale for the chosen software architecture.
- **SWE.2.BP7: Establish bidirectional traceability.** Establish bidirectional traceability between software requirements and elements of the software architectural design.
- **SWE.2.BP8: Ensure consistency.** Ensure consistency between software requirements and the software architectural design.
- **SWE.2.BP9: Communicate agreed software architectural design.** Communicate the agreed software architectural design and updates to software architectural design to all relevant parties.

04-04 Software architectural design

- overall software structure
- operative system including task structure
- Identifies inter-task/inter-process communication
- the required software elements
- own developed and supplied code
- the relationship and dependency between software elements
- where the data (such as parameters) are stored and which measures (e.g. checksums, redundancy) are taken to prevent data corruption
- variants for different model series or configurations are derived
- dynamic behavior of the software (Start-up, shutdown, software update, error handling and recovery, etc.)
- which data is persistent and under which conditions
- Consideration is given to:
  - any required software performance characteristics
  - any required software interfaces
  - any required security characteristics required
  - any database design requirements

Traceability requirements

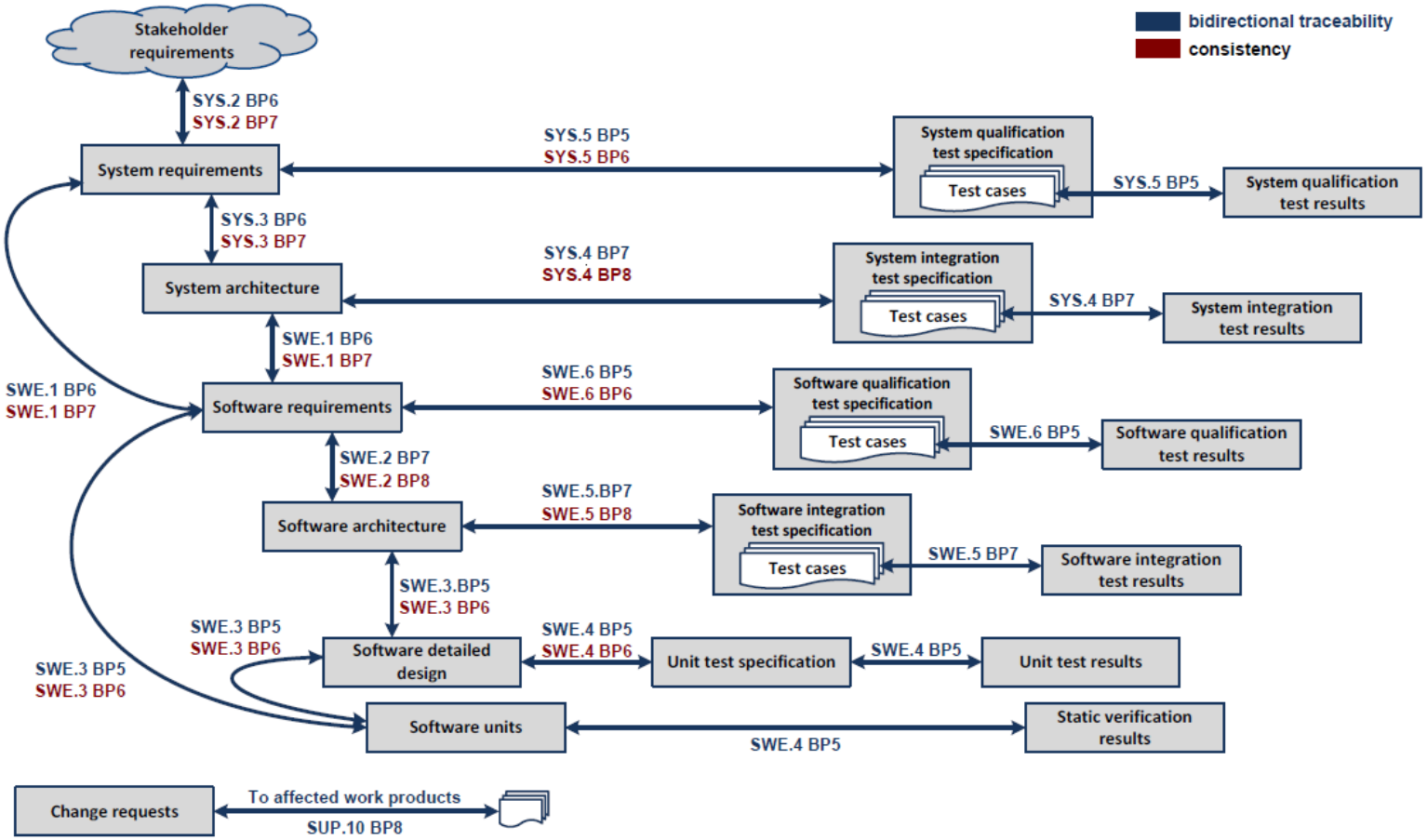
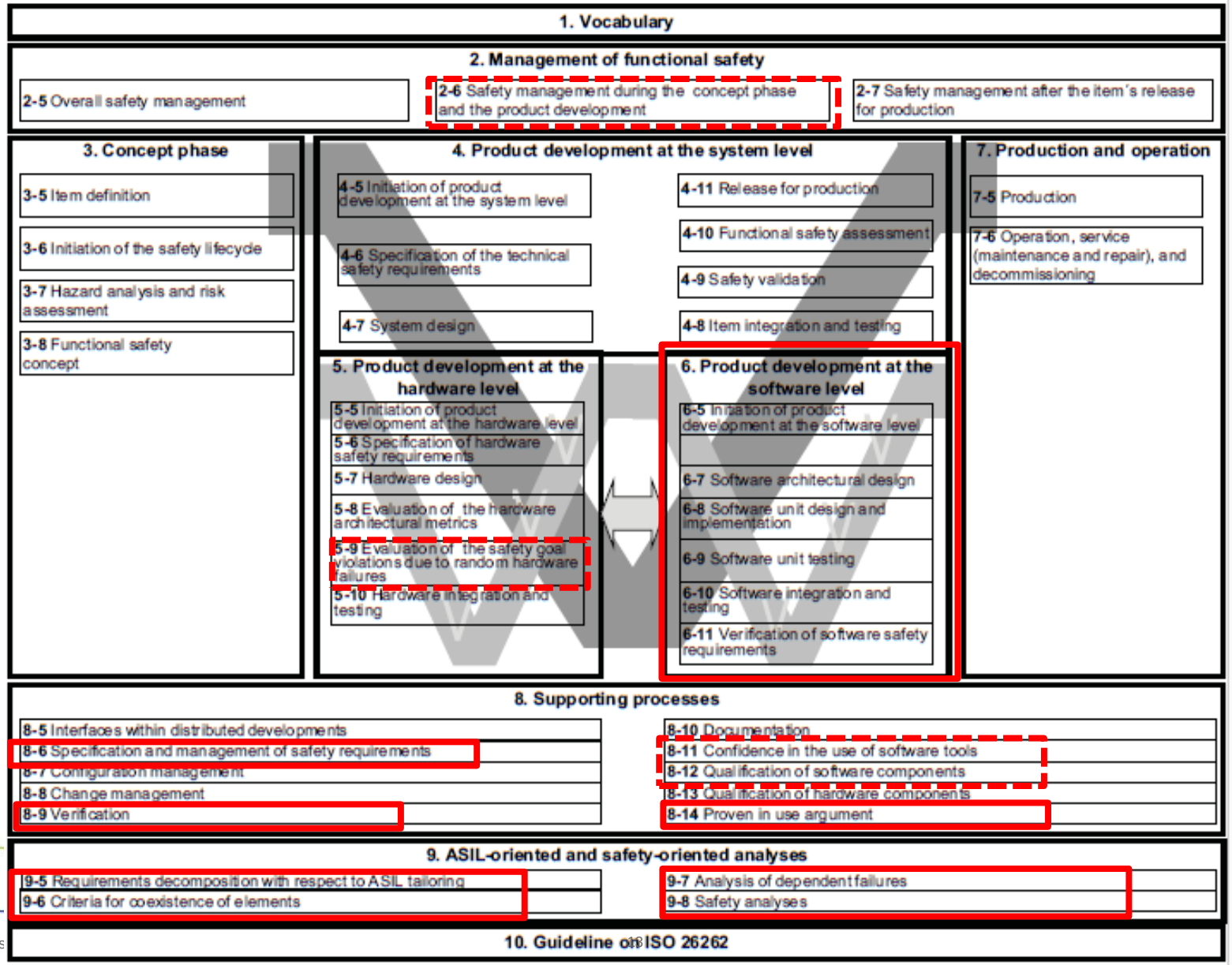


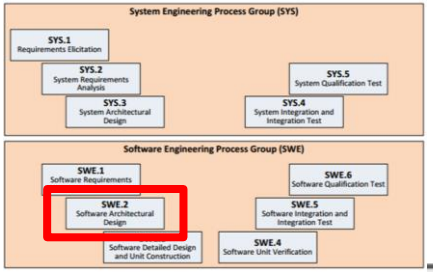
Figure D.4 — Bidirectional Traceability and Consistency

addalot QUALITY IMPROVEMENT

The whole 26262 – SW impacts

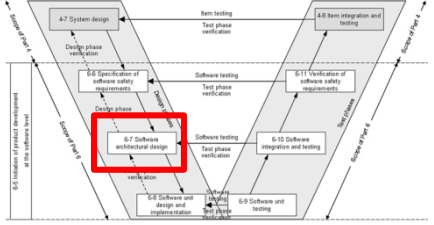


Software level



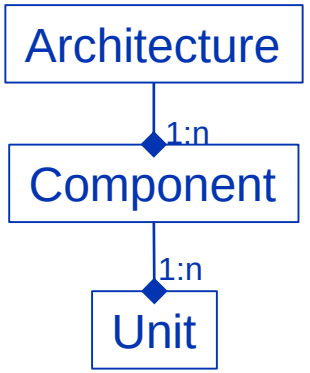
addalot QUALITY IMPROVEMENT

7: Software architectural design



Objectives

1. Develop a software architectural design that realizes the software safety requirements
2. Verify the software architectural design



General

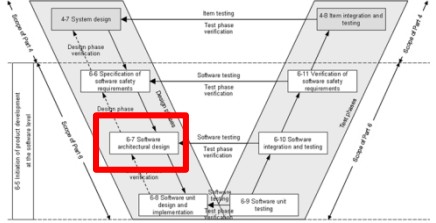
The software architectural design represents all software components and their interactions in a hierarchical structure. **Static aspects**, such as interfaces and data paths between all software components, as well as **dynamic aspects**, such as process sequences and timing behaviour are described.

- NOTE The software architectural design is not necessarily limited to one microcontroller or ECU, and is related to the technical safety concept and system design. The software architecture for each microcontroller is also addressed by this chapter.

- In order to develop a software architectural design both software safety requirements as well as all non-safety-related requirements are implemented.
- The software architectural design provides the means to implement the software safety requirements and to manage the complexity of the software development.

addalot QUALITY IMPROVEMENT

7: Prerequisites



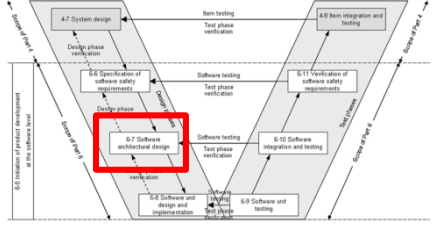
- safety plan (refined) in accordance with 5.5.1;
- design and coding guidelines for modelling and programming languages in accordance with 5.5.3;
- hardware-software interface specification in accordance with ISO 26262-4:2011, 7.5.3;
- software safety requirements specification in accordance with 6.5.1;
- software verification plan (refined) in accordance with 6.5.3; and
- software verification report in accordance with 6.5.4.

Support information

- technical safety concept (see ISO 26262-4:2011, 7.5.1);
- system design specification (see ISO 26262-4:2011, 7.5.2);
- qualified software components available (see ISO 26262-8:2011, Clause 12);
- tool application guidelines in accordance with 5.5.4; and
- guidelines for the application of methods (from external source).

addalot QUALITY IMPROVEMENT

Requirements and recommendations



7.4.1 Use of appropriate notation

7.4.2 Design considerations

7.4.3 Design principles

7.4.4 Identification of software units

7.4.5 Design aspects

7.4.6 Component categorization

7.4.7 New/modified components

7.4.8 Re-used components

7.4.9 Allocation of Safety requirements

7.4.10 ASIL of combined components

7.4.11 Software partitioning (Annex D)

7.4.12 Dependent failure analysis (Part 9: 7 Dependent failure analysis)

7.4.13 Safety analysis (Part 9: 8 Safety analysis)

7.4.14 Error detection

7.4.15 Error handling

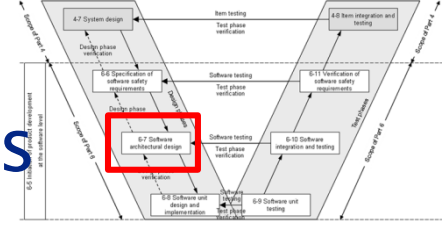
7.4.16 New hazards

7.4.17 Resource usage

7.4.18 Architectural design verification

addalot QUALITY IMPROVEMENT

7.4.1 Notation and 7.4.2 Considerations



7.4.1 Notations for software architectural design

	Methods (Notations)	A	B	C	D
1a	Informal	++	++	+	+
1b	Semi-formal (also executable models)	+	++	++	++
1c	Formal	+	+	+	+

Informal: Power-point drawings

Semi-formal: UML, SDL or UML-RT, **An overview** of architecture design notations

7.4.2 Design considerations: During the development of the software architectural design the following shall be considered:

a. the verifiability of the software architectural design;

   NOTE This implies bi-directional traceability between the software architectural design and the **software safety requirements**. (only safety requirements)

b. the suitability for configurable software;

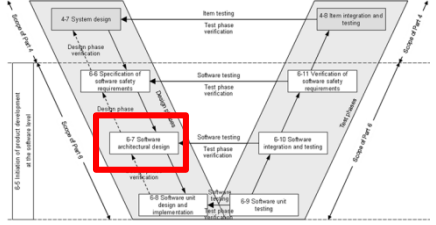
c. the feasibility for the design and implementation of the software units;

d. the testability of the software architecture during software integration testing; and

e. the maintainability of the software architectural design.

addalot QUALITY IMPROVEMENT

7.4.3 Design principles



7.4.3 Principles of software architecture design In order to avoid failures resulting from high complexity, the software architectural design shall exhibit the following properties by use of the principles listed in Table below:

a. modularity;

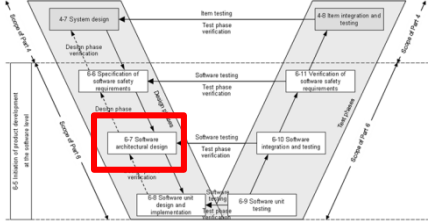
b. encapsulation, and

c. simplicity.

	Methods	A	B	C	D
1a	Hierarchical structure of software components	++	++	++	++
1b	Restricted size of software components	++	++	++	++
1c	Restricted size of interfaces	+	+	+	+
1d	High cohesion within each software component	+	++	++	++
1e	Restricted coupling between software components	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts (must have priority)	+	+	+	++

addalot QUALITY IMPROVEMENT

7.4.4 Level and 7.4.5 Description



7.4.4 The software architectural design shall be developed down to the **level** where all software units are identified.

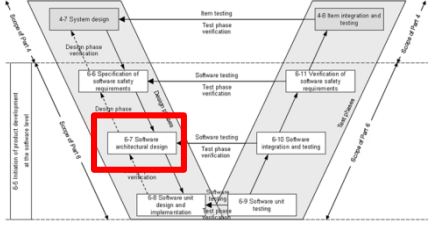
7.4.5 The software architectural design shall **describe**:

a. the static design aspects of the software components; i.e.
   - the software structure including its hierarchical levels;
   - the logical sequence of data processing;
   - the data types and their characteristics;
   - the external interfaces of the software components;
   - the external interfaces of the software; and
   - the constraints including the scope of the architecture and external dependencies.

b. the dynamic design aspects of the software components, i.e.
   - the functionality and behaviour; (Note 2: including operating states e.g. power-up, shut-down, normal operation, calibration and diagnosis)
   - the control flow and concurrency of processes; (Note 3: including allocation to HW)
   - the data flow between the software components;
   - the data flow at external interfaces; and
   - the temporal constraints.

addalot QUALITY IMPROVEMENT

7.4.6-7.4.8 Reuse categorization



7.4.6 Every safety-related software component shall be categorized as one of the following:

a. newly developed;

b. reused with modifications; or

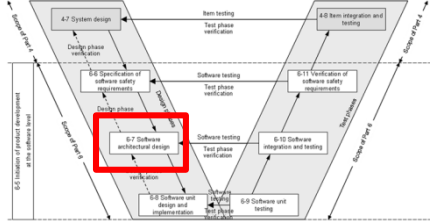
c. reused without modifications.

7.4.7 Safety-related software components that are newly developed or reused with modifications shall be developed in accordance with ISO 26262.

7.4.8 Safety-related software components that are reused without modifications shall be qualified in accordance with ISO 26262-8:2011, Clause 12.

addalot QUALITY IMPROVEMENT

7.4.9 Req allocation and ASIL

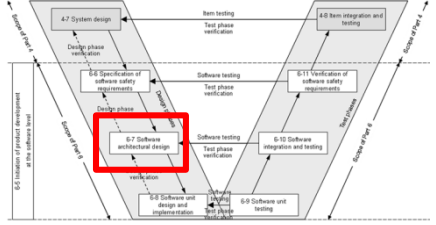


7.4.9 The software safety requirements shall be allocated to the software components. As a result, each software component shall be developed in compliance with the highest ASIL of any of the requirements allocated to it.

- NOTE Following this allocation, further refinement of the software safety requirements can be necessary.

Software components = one or more Software Units, thus we need to allocate safety reqs close to software units

7.4.10 Highest ASIL or no interference



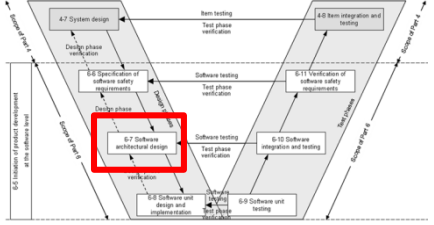
7.4.10 If the embedded software has to implement software components of **different ASILs**, or safety-related and non-safety-related software components, then all of the embedded software shall be treated in accordance with the **highest ASIL**, unless the software components meet the criteria for coexistence in accordance with ISO 26262-9:2011, Clause 6.

Freedom of interference general in ISO 26262-9:2011, Clause 6.

- This means that cascading failures from this sub-element to the safety-related elements are absent.
- This can be achieved by design precautions such as those concerning the data flow and control flow for software, or the I/O signals and control lines for hardware.

Software specific in Annex D (See separate slides)

7.4.11 Software Partitioning



7.4.11 If **software partitioning** (see Annex D) is used to implement freedom from interference between software components it shall be ensured that:

a) the **shared resources** are used in such a way that freedom from interference of software partitions is ensured;
   - NOTE 1 Tasks within a software partition are not free from interference among each other.
   - NOTE 2 One software partition cannot change the code or data of another software partition nor command non-shared resources of other software partitions.
   - NOTE 3 The service received from shared resources by one software partition cannot be affected by another software partition. This includes the performance of the resources concerned, as well as the rate, latency, jitter and duration of scheduled access to the resource.

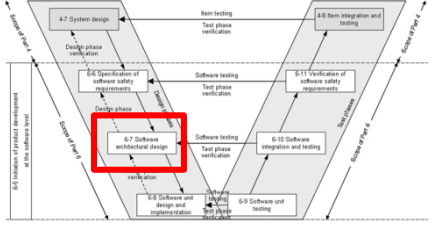
b) the software partitioning is **supported** by dedicated **hardware** features or equivalent means (this requirement applies to ASIL D, in accordance with 4.3);

c) the part of the software that implements the software partitioning is developed in compliance with the same or an ASIL higher than the highest ASIL assigned to the requirements of the software partitions; and
   - NOTE In general the operating system provides or supports software partitioning.

d) the **verification** of the software partitioning during software integration and testing (in accordance with Clause 10) is performed.

addalot QUALITY IMPROVEMENT

7.4.12 Dependent failures



7.4.12 An analysis of dependent failures in accordance with ISO 26262-9:2011, Clause 7, shall be carried out if the implementation of software safety requirements relies on freedom from interference or sufficient independence between software components.

Part 9: Clause 7: Analysis of dependent failures

Objective: The analysis of dependent failures aims to identify the single events or single causes that could bypass or invalidate a required independence or freedom from interference between given elements and violate a safety requirement or a safety goal.

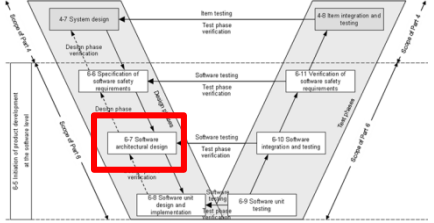
Architectural features to consider:

- similar and dissimilar redundant elements;
- different functions implemented with identical software or hardware elements;
- functions and their respective safety mechanisms;
- partitions of functions or software elements;
- physical distance between hardware elements, with or without barrier;
- common external resources.

- Independence is threatened by common cause failures and cascading failures, while freedom from interference is only threatened by cascading failures.

(Detour)

7.4.13 Safety analysis

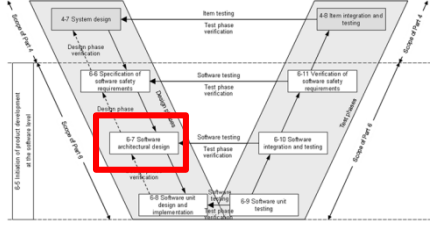


7.4.13 Safety analysis shall be carried out at the software architectural level in accordance with ISO 26262-9:2011, Clause 8, in order to: (next)

- identify or confirm the safety-related parts of the software; and
- support the specification and verify the efficiency of the safety mechanisms.

NOTE Safety mechanisms can be specified to cover both issues associated with random hardware failures as well as software faults.

7.4.14 Error detection

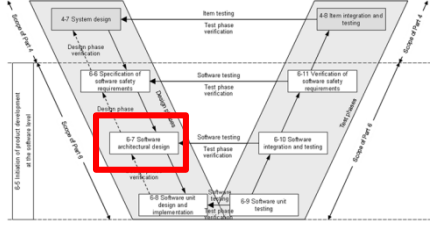


7.4.14 To specify the necessary software safety mechanisms at the software architectural level, based on the results of the safety analysis in accordance with 7.4.13, mechanisms for **error detection** as listed in Table 4 shall be applied.

- NOTE When not directly required by technical safety requirements allocated to software, the use of software safety mechanisms is reviewed at the system level to analyse the potential impact on the system behaviour.

	Mechanisms for error detection	A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check (reference model, comparing sources)	+	+	+	++
1c	Detection of data errors (EDC, multiple storage)	+	+	+	+
1d	External monitoring facility (watchdog)	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

7.4.15 Error handling

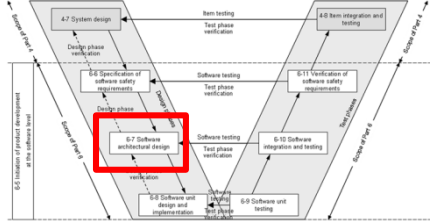


7.4.15 This subclause applies to ASIL (A), (B), C and D, in accordance with 4.3 ((X) = recommendation): to specify the necessary software safety mechanisms at the software architectural level, based on the results of the safety analysis in accordance with 7.4.13, mechanisms for **error handling** as listed in Table 5 shall be applied.

- NOTE 1 When not directly required by technical safety requirements allocated to software, the use of software safety mechanisms is reviewed at the system level to analyse the potential impact on the system behaviour.
- NOTE 2 The analysis at software architectural level of possible hazards due to hardware is described in ISO 26262-5.

	Mechanisms for error handling	A	B	C	D
1a	Static recovery mechanism (forward and backward, blocks, repetition = reset HW and re-execute SW)	+	+	+	+
1b	Graceful degradation (prioritizing functions)	+	+	++	++
1c	Independent parallel redundancy	o	o	+	++
1d	Correcting codes for data	+	+	+	+

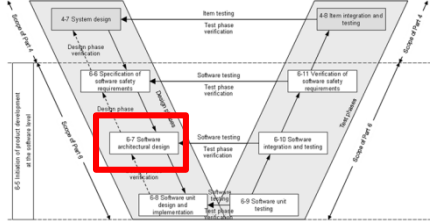
7.4.16 New hazards



7.4.16 If **new hazards** introduced by the software architectural design are not already covered by an existing safety goal, they shall be introduced and evaluated in the hazard analysis and risk assessment in accordance with the change management process in ISO 26262-8:2011, Clause 8.

- NOTE Newly identified hazards, not already reflected in a safety goal, are usually non-functional hazards. If those non-functional hazards are outside the scope of this standard then it is recommended that they be annotated in the hazard analysis and risk assessment with the following statement “No ASIL is assigned to this hazard as it is not within the scope of ISO 26262.” However, an ASIL is allowed for reference purposes.

7.4.17 Resource usage



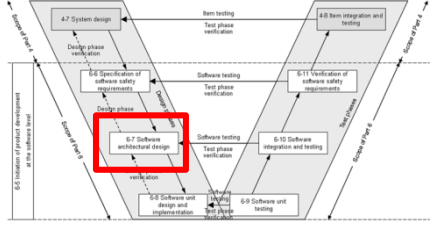
7.4.17 An upper estimation of required **resources** for the embedded software shall be made, including:

a) the execution time;

b) the storage space; and
EXAMPLE RAM for stacks and heaps, ROM for program and non-volatile data.

c) the communication resources.

7.4.18 Verification

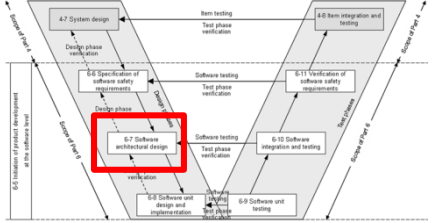


7.4.18 The software architectural design shall be **verified** in accordance with ISO 26262-8:2011, Clause 9, and by using the software architectural design verification methods listed in Table 6 to demonstrate the following properties:

- compliance with the software safety requirements;
- compatibility with the target hardware; and
  - NOTE This includes the resources as specified in 7.4.17.
- adherence to design guidelines.

	Methods for SW architecture design verification	A	B	C	D
1a	Walk-through of the design	++	+	o	o
1b	Inspection of the design	+	++	++	++
1c	Simulation of dynamic parts of the design	+	+	+	++
1d	Prototype generation	o	o	+	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis	+	+	++	++
1g	Data flow analysis	+	+	++	++

7: Work products



- **7.5.1 Software architectural design specification** resulting from requirements 7.4.1 to 7.4.6, 7.4.9, 7.4.10, 7.4.14, 7.4.15 and 7.4.17.
- **7.5.2 Safety plan (refined)** resulting from requirement 7.4.7.
- **7.5.3 Software safety requirements specification (refined)** resulting from requirement 7.4.9.
- **7.5.4 Safety analysis report** resulting from requirement 7.4.13.
- **7.5.5 Dependent failures analysis report** resulting from requirement 7.4.12.
- **7.5.6 Software verification report (refined)** resulting from requirement 7.4.18.

ASPICE Output work products
04-04 Software architectural design
04-04 Software architectural design
04-04 Software architectural design 17-08 Interface requirement specification
04-04 Software architectural design
04-04 Software architectural design 13-19 Review record 13-22 Traceability record
13-04 Communication record

addalot QUALITY IMPROVEMENT

Comparison

ASPICE	26262
SWE.2.BP1: Develop software architectural design	7.4.1 Notations for software architectural design 7.4.3 Principles of software architecture design 7.4.4 Required level 7.4.5 Description (a: static)
- (own and supplied code – Arch doc)	7.4.6-7.4.8 ASIL categorize of components (re-used)
- (data consistency mechanisms – Arch doc)	7.4.14 and 7.4.15 Software safety mechanisms
-	7.4.16 Analyze new hazards
SWE.2.BP2: Allocate software requirements	7.4.9 Allocate software **safety** requirements
-	7.4.10 ASIL analysis of components
- (dependency analysis – Arch doc)	7.4.11 Software partitioning 7.4.12 Dependent failure analysis
SWE.2.BP3: Define interfaces of software elements	7.4.5 Description (a: Interfaces)
SWE.2.BP4: Describe dynamic behavior	7.4.5 Description (b: dynamic)
SWE.2.BP5: Define resource consumption objectives	7.4.17 Resource limits
SWE.2.BP6: Evaluate alternative software architectures	- (7.4.2 Design considerations)
SWE.2.BP7: Establish bidirectional traceability	7.4.2 Design considerations (a) 7.4.9 Allocate software **safety** requirements
SWE.2.BP8: Ensure consistency	7.4.18 Verify compliance, 7.4.2 Design considerations
-	7.4.13 Safety analysis
SWE.2.BP9: Communicate agreed software architectural design	-

“Excellent firms don't believe in excellence - only in constant improvement and change.”

In Search of Excellence - Tom Peters

Even-Andre.Karlsson@addalot.se
+46 706 800 533

addalot
QUALITY IMPROVEMENT