[4 Requirements for compliance 2](#_Toc12111617)

[5 Initiation of product development at the software level 2](#_Toc12111618)

[6 Specification of software safety requirements 6](#_Toc12111619)

[7 Software architectural design 8](#_Toc12111620)

[8 Software unit design and implementation 19](#_Toc12111621)

[9 Software unit testing 23](#_Toc12111622)

[10 Software integration and testing 27](#_Toc12111623)

[11 Verification of software safety requirements 33](#_Toc12111624)

[Annex D (informative) 40](#_Toc12111625)

ISO26262-6

This part of ISO 26262 specifies the requirements for product development at the software level for automotive applications, including the following:

1. requirements for initiation of product development at the software level,
2. specification of the software safety requirements,
3. software architectural design,
4. software unit design and implementation,
5. software unit testing,
6. software integration and testing, and
7. verification of software safety requirements.

# 4 Requirements for compliance

## 4.1 General requirements

When claiming compliance with ISO 26262, each requirement shall be complied with, unless one of the following applies:

a) tailoring of the safety activities in accordance with ISO 26262-2 has been planned and shows that the requirement does not apply, or

b) a rationale is available that the non-compliance is acceptable and the rationale has been assessed in accordance with ISO 26262-2.

Information marked as a “NOTE” or “EXAMPLE” is only for guidance in understanding, or for clarification of the associated requirement, and shall not be interpreted as a requirement itself or as complete or exhaustive.

**WORK PRODUCTS** The results of safety activities are given as work products.

**PREREQUISITES** “Prerequisites” are information which shall be available as work products of a previous phase.

Given that certain requirements of a clause are ASIL-dependent or may be tailored, certain work products may not be needed as prerequisites.

“Further supporting information” is information that can be considered, but which in some cases is not required by ISO 26262 as a work product of a previous phase and which may be made available by external sources that are different from the persons or organizations responsible for the functional safety activities.

## 4.3 ASIL-dependent requirements and recommendations

The requirements or recommendations of each subclause shall be complied with for ASIL A, B, C and D, if not stated otherwise.

These requirements and recommendations refer to the ASIL of the safety goal.

If ASIL decomposition has been performed at an earlier stage of development, in accordance with ISO 26262-9:2011, Clause 5, the ASIL resulting from the decomposition shall be complied with.

If an ASIL is given in parentheses in ISO 26262, the corresponding subclause shall be considered as a recommendation rather than a requirement for this ASIL. This has no link with the parenthesis notation related to ASIL decomposition.

# 5 Initiation of product development at the software level

## 5.2 General

**METHODS (SUPORTED BY GUIDELINES AND TOOLS) DETERMINED AND PLANNED** The initiation of the *software development* is a planning activity, where *software development* sub-phases and their supporting processes (see ISO 26262-8 and ISO 26262-9) are **determined** and **planned** according to the extent and complexity of the item development.

The *software development* sub-phases and supporting processes are initiated by determining the appropriate methods in order to comply with the requirements and their respective ASIL.

The methods are supported by guidelines and tools, which are determined and planned for each sub-phase and supporting process.

NOTE Tools used for *software development* can include tools other than software tools.

EXAMPLE Tools used for testing phases.

The planning of the *software development at the software level*  includes the coordination with the *product development at the system level* (see ISO 26262-4) and the *hardware level* (see ISO 26262-5).

## 5.3 Inputs to this clause

### 5.3.1 Prerequisites

The following information **shall be** **available** **(ALL FROM ISO 26262-4)**

|  |  |  |
| --- | --- | --- |
| Requirement | Question | Answer (Y/N) |
| project plan (refined) in accordance with ISO 26262 4:2011, 5.5.1; | Is the project plan (refined) available? |  |
| safety plan (refined) in accordance with ISO 26262 4:2011, 5.5.2; | Is the safety plan (refined) available? |  |
| technical safety concept in accordance with ISO 26262 4:2011, 7.5.1; | Is the technical safety concept available? |  |
| system design specification in accordance with ISO 26262 4:2011, 7.5.2; and | Is the system design specification available? |  |
| item integration and testing plan (refined) in accordance with ISO 26262 4:2011, 8.5.1. | Is the item integration and testing plan (refined) available? |  |

### 5.3.2 Further supporting information

The following information can be considered:

1. qualified software tools available (see ISO 26262-8:2011, Clause 11);
2. qualified software components available (see ISO 26262-8:2011, Clause 12);
3. design and coding guidelines for modelling and programming languages (from external source);
4. guidelines for the application of methods (from external source); and
5. guidelines for the application of tools (from external source).

## 5.4 Requirements and recommendations

|  |  |  |  |
| --- | --- | --- | --- |
| Requirements | Question | (Y/N) | Work Product |
| **5.4.1** The activities and the determination of appropriate methods *for the product development at the software level* **shall be planned**. | Were the activities and the determination of appropriate methods *for the product development at the software level,* planned? |  | **5.5.1 Safety plan (refined)**  **5.5.2 Software verification plan** |
| **5.4.2** The tailoring of the lifecycle *for product development at the software level* **shall be performed** in accordance with ISO 26262-2:2011, 6.4.5, and based on the reference phase model given in Figure 2. | Was the tailoring of the lifecycle *for product development at the software level,* performed? |  | **5.5.1 Safety plan (refined)**  **5.5.2 Software verification plan** |
| **5.4.3** If developing configurable software, Annex C **shall be applied**. | Is software configurable?  If Yes, was Annex C applied? |  | **5.5.1 Safety plan (refined)**  **5.5.2 Software verification plan** |

|  |  |  |  |
| --- | --- | --- | --- |
| **5.4.4** The *software development* process for the software of an item, including lifecycle phases, methods, languages and tools, **shall be consistent** across all the sub-phases of the software lifecycle and | Were the methods, languages, lifecycle and tools consistent across all the sub-phases of the software development process? |  | **5.5.1 Safety plan (refined)**  **5.5.2 Software verification plan** |
| **shall be compatible** with the system and hardware development phases, such that the required data can be transformed correctly.  NOTE The sequencing of phases, tasks and activities, including iteration steps, for the software of an item is to ensure the consistency of the corresponding work products with the product development at the hardware level (see ISO 26262-5) and the product development at the system level (see ISO 26262-4). | Were the system and hardware development phase compatible with the software development phase? (such that the required data can be transformed correctly) ? |  |  |
| **5.4.5** For each sub-phase of *software development*, the selection of the following, including guidelines for their application, **shall be carried out**:  a) Methods;  b) Corresponding tools. | Were methods and the corresponding tools selection, including guidelines, carried out? |  | **5.5.1 Safety plan (refined)**  **5.5.4 Tool application guidelines** |

|  |  |  |  |
| --- | --- | --- | --- |
| **5.4.6** The criteria that **shall be considered** when selecting a suitable modelling or programming language are: | Were the criteria when selecting a suitable modelling or programming language: |  | **5.5.1 Safety plan (refined)**  **5.5.3 Design and coding guidelines for modelling and programming languages**  **5.5.4 Tool application guidelines** |
| a) an unambiguous definition;  EXAMPLE Syntax and semantics of the language. | Unambiguously defined? |  |  |
| b) the support for embedded real time software and runtime error handling; | Has support for embedded real time software and runtime error handling? |  |  |
| c) the support for modularity, abstraction and structured constructs | Has support for modularity, abstraction and structured constructs? |  |  |
| Criteria that are not sufficiently addressed by the language itself shall be covered (enforced) by the corresponding **guidelines**, or by the **development environment**. |  |  |  |
| NOTE 1 The selected programming language (such as ADA, **C**, C++, Java, Assembler or a graphical modelling language) supports the topics given in 5.4.7. **Programming or modelling guidelines** can be used to comply with these topics.  NOTE 2 Assembly languages can be used for those parts of the software where the use of high-level programming languages is not appropriate, such as low-level software with interfaces to the hardware, interrupt handlers, or time-critical algorithms. |  |  |  |
| **5.4.7** To support the correctness of the design and implementation, the design and coding guidelines for the modelling, or programming languages, **shall address the topics listed in Table 1.**  NOTE 1 Coding guidelines are usually different for different programming languages.  NOTE 2 Coding guidelines can be different for model-based development.  NOTE 3 Existing coding guidelines can be modified for a specific item development.  EXAMPLE MISRA C[3] and MISRA AC AGC[4] are coding guidelines for the programming language C. | Were the design and coding guidelines for the modelling, or programming languages addressed in accordance with Table 1? |  | **5.5.1 Safety plan (refined)**  **5.5.2 Software verification plan**  **5.5.3 Design and coding guidelines for modelling and programming languages** |

#### Table 1 — Topics to be covered by modelling and coding guidelines

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Topics** | | ASIL | | | |  |
| A | B | C | D |  |
| 1a | Enforcement of low complexity a | ++ | ++ | ++ | ++ | not enforced |
| 1b | Use of language subsets b | ++ | ++ | ++ | ++ | MISRA C used |
| 1c | Enforcement of strong typing c | ++ | ++ | ++ | ++ | With PCLint |
| 1d | Use of defensive implementation techniques | o | + | ++ | ++ | not applied |
| 1e | Use of established design principles | + | + | + | ++ | not defined |
| 1f | Use of unambiguous graphical representation | + | ++ | ++ | ++ | MS Visio |
| 1g | Use of style guides | + | ++ | ++ | ++ | Coding conventions |
| 1h | Use of naming conventions | ++ | ++ | ++ | ++ | Coding conventions |

a An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required.

b The objectives of method 1b are

Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.

Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.

Exclusion of language constructs which could result in unhandled run-time errors.

c The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.

# 6 Specification of software safety requirements

|  |  |
| --- | --- |
| Objectives |  |
| The first objective of this sub phase is to specify the software safety requirements (Requirements related to software) They are derived from the technical safety concept and the system design specification. | Are the software safety requirements specified based on the Technical safety concept and the system design specification? |
| The second objective is to detail the hardware-software interface requirements initiated in ISO 26262-4:2011,  Clause 7. | Are the hardware-software interface requirements detailed? |
| The third objective is to verify that the software safety requirements and the hardware-software interface requirements are consistent with the technical safety concept and the system design specification. | Are the software safety requirements and the hardware-software interface requirements consistent with the technical safety concept and the system design specification? |

**6.3 Inputs to this clause**

**6.3.1 Prerequisites**

The following information shall be available:

 technical safety concept in accordance with ISO 26262-4:2011, 7.5.1;

 system design specification in accordance with ISO 26262-4:2011, 7.5.2;

 hardware-software interface specification in accordance with ISO 26262-4:2011, 7.5.3;

 safety plan (refined) in accordance with 5.5.1;

 software verification plan in accordance with 5.5.2.

**6.3.2 Further supporting information**

The following information can be considered:

 hardware design specification (see ISO 26262-5:2011, 7.5.1);

 guidelines for the application of methods (from external source).

**6.4 Requirements and recommendations**

|  |  |
| --- | --- |
| **6.4.1** The software safety requirements **shall address** each software-based function whose failure could lead to a violation of a technical safety requirement allocated to software.  EXAMPLE Functions whose failure could lead to a violation of a safety requirement can be:  - functions that enable the system to achieve or maintain a safe state;  - functions related to the detection, indication and handling of faults of safety-related hardware elements;  - functions related to the detection, notification and mitigation of faults in the software itself;  NOTE 1 These include both the self-monitoring of the software in the operating system and application-specific self-monitoring of the software to detect, indicate and handle systematic faults in the application software.  - functions related to on-board and off-board tests;  NOTE 2 On-board tests can be carried out by the system itself or through other systems within the vehicle network during operation and during the pre-run and post-run phase of the vehicle.  NOTE 3 Off-board tests refer to the testing of the safety-related functions or properties during production or in service.  - functions that allow modifications of the software during production and service; and  - Functions related to performance or time-critical operations. | Arefunctions that enable the system to achieve or maintain a safe state addressed?  Are functions related to the detection, indication and handling of faults of safety-related hardware elements addressed?  Are functions related to the detection, notification and mitigation of faults in the software itself?  Are functions that allow modifications of the software during production and service addressed?  Are functions related to performance or time-critical operations addressed?  Shall produce a Work Product **Software safety requirements specification** |
| 6.4.2 The specification of the software safety requirements shall be derived from the technical safety concept and the system design in accordance with ISO 26262-4:2011, 7.4.1 and 7.4.5, and shall consider:  a) the specification and management of safety requirements in accordance with ISO 26262-8:2011,Clause 6;  b) the specified system and hardware configurations;  EXAMPLE 1 Configuration parameters can include gain control, band pass frequency and clock prescaler.  c) the hardware-software interface specification;  d) the relevant requirements of the hardware design specification;  e) the timing constraints;  EXAMPLE 2 Execution or reaction time derived from the required response time at the system level.  f) the external interfaces; and  EXAMPLE 3 Communication and user interfaces.  g) each operating mode of the vehicle, the system, or the hardware, having an impact on the software.  EXAMPLE 4 Operating modes of hardware devices can include default, initialization, test, and advanced modes. | Was the specification of the Software Safety Requirements derived from the Technical Safety Concept and from the System Design Specification? |
| **6.4.3** If ASIL decomposition is applied to the software safety requirements, ISO 26262-9:2011, Clause 5,  shall be complied with. | Shall produce a Work Product **Software safety requirements specification** |
| **6.4.4** The hardware-software interface specification initiated in ISO 26262-4:2011, Clause 7, **shall be** detailed down to a level allowing the correct control and usage of hardware, and shall describe each safety-related dependency between hardware and software. | Shall produce a Work Product **Hardware-software interface specification (refined)** |
| **6.4.5** If other functions in addition to those functions for which safety requirements are specified in 6.4.1 are  carried out by the embedded software, these functions **shall be specified**, or else a reference made to their  specification. | Shall produce a Work Product **Software safety requirements specification** |
| **6.4.6** The verification of the software safety requirements and the verification of the refined specification of the hardware software interface shall be planned in accordance with ISO 26262-8:2011, Clause 9. | Shall produce a Work Product **Software verification plan (refined)** |
| **6.4.7** The refined hardware-software interface specification shall be verified jointly by the persons responsible for the system, hardware and software development. | Shall produce a Work Product **Software verification report** |
| **6.4.8** The software safety requirements and the refined hardware-software interface requirements shall be  verified in accordance with ISO 26262-8:2011, Clauses 6 and 9, to show their:  a) compliance and consistency with the technical safety requirements;  b) compliance with the system design; and  c) consistency with the hardware-software interface. | Shall produce a Work Product **Software verification report** |
|  |  |
|  |  |
|  |  |

**6.5 Work products**

**6.5.1 Software safety requirements specification** resulting from requirements 6.4.1 to 6.4.3 and 6.4.5.

**6.5.2 Hardware-software interface specification (refined)** resulting from requirement 6.4.4.

NOTE This work product refers to the same work product as given in ISO 26262-5:2011 6.5.2

**6.5.3 Software verification plan (refined)** resulting from requirement 6.4.6.

**6.5.4 Software verification report** resulting from requirements 6.4.7 and 6.4.8.

# 7 Software architectural design

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 behavior 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.**

Hence in this sub-phase safety-related and non-safety-related requirements are handled within one development process.

The software architectural design provides the means to implement the software safety requirements and to manage the complexity of the software development.

## 7.1 Objectives

|  |  |
| --- | --- |
| The first objective of this sub phase is to develop a *software architectural design* that realizes the software safety requirements. |  |
| The second objective of this sub phase is to verify the *software architectural design*. |  |

## 7.4 Requirements and recommendations

|  |  |  |
| --- | --- | --- |
|  |  | Comment/Answer |
| **7.4.1 ABSTRACTION** To ensure that the *software architectural design* captures the information necessary to allow the subsequent development activities to be performed correctly and effectively, the *software architectural design* **shall be described** with appropriate levels of abstraction by using the notations for software architectural design listed in Table 2. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| Notation used to describe the software architectural design: |  |
| Informal notations |  |
| Semi-formal notations |  |
| Formal notations |  |
| **7.4.2** During the development of the *software architectural design* the following **shall be considered**:  a) **VERIFIABILITY** the verifiability of the software architectural design;  NOTE This implies bi-directional traceability between the software architectural design and the software safety requirements.  b) **SUITABILITY** the suitability for configurable software;  c) **FEASIBILITY** the feasibility for the design and implementation of the software units;  d) **TESTABILITY** the testability of the software architecture during software integration testing; and  e) **MAINTAINABILITY** the maintainability of the software architectural design. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| a) Was traceability between the software architectural design and the software safety requirements used? |  |
| b) If configurable software is used, was the software architectural design suit? |  |
| c) Is the design and implementation of the software units feasible? |  |
| d) Can the software architecture be tested during software integration testing? |  |
| e) Is software architectural design maintainable? |  |
| **7.4.3** 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 3:  a) **MODULARITY** modularity;  b) **ENCAPSULATION** encapsulation; and  c) **SIMPLICITY** simplicity.  NOTE An appropriate compromise between the methods listed in Table 3 can be necessary since the methods are not mutually exclusive. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| Does the software architectural design exhibit |  |
| a) modularity; |  |
| b) encapsulation; and |  |
| c) simplicity? |  |

### Table 3 — Principles for software architectural design

<http://publications.lib.chalmers.se/records/fulltext/184469/184469.pdf>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Methods | | ASIL | | | | Status |
| A | B | C | D |  |
| 1a | Hierarchical structure of software components | ++ | ++ | ++ | ++ | Need to be introduced |
| 1b | Restricted size of software components a | ++ | ++ | ++ | ++ | Taken into consideration |
| 1c | Restricted size of interfaces a | + | + | + | + | No restrictions are implied |
| 1d | High cohesion within each software component b | + | ++ | ++ | ++ | Taken into consideration |
| 1e | Restricted coupling between software components a,b,c | + | ++ | ++ | ++ | Analyzed with “SourceMonitor” |
| 1f | Appropriate scheduling properties | ++ | ++ | ++ | ++ | Needs improvement |
| 1g | Restricted use of interrupts a,d | + | + | + | ++ | Procedure needs improvemen |

a In methods 1b, 1c, 1e and 1g "restricted" means to minimize in balance with other design considerations.

b Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.

c Method 1e addresses the limitation of the external coupling of software components.

d Any interrupts used have to be priority-based

|  |  |  |
| --- | --- | --- |
| **7.4.4 SOFTWARE UNITS** The software architectural design **shall be developed** down to the level where all software units are identified. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| Are all software units identified? |  |
| **7.4.5** The software architectural design **shall describe**:  a) **STATIC DESIGN** the static design aspects of the software components; and  NOTE 1 Static design aspects address:  - 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.  NOTE 2 In the case of model-based development, modelling the structure is an inherent part of the overall modelling activities.  b) **DYNAMIC DESIGN** the dynamic design aspects of the software components.  NOTE 1 Dynamic design aspects address:  - the functionality and behavior;  - the control flow and concurrency of processes;  - the data flow between the software components;  - the data flow at external interfaces; and  - the temporal constraints.  NOTE 2 To determine the dynamic behavior (e.g. of tasks, time slices and interrupts) the different operating states (e.g. power-up, shut-down, normal operation, calibration and diagnosis) are considered.  NOTE 3 To describe the dynamic behavior (e.g. of tasks, time slices and interrupts) the communication relationships and their allocation to the system hardware (e.g. CPU and communication channels) are specified. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| In the Static design, Is the software structure including its hierarchical levels described? |  |
| In the Static design, Is the logical sequence of data processing described? |  |
| In the Static design, Is the data types and their characteristics described? |  |
| In the Static design, Is the external interfaces of the software components described? |  |
| In the Static design, Is the external interfaces of the software described? |  |
| In the Static design, Is the constraints including the scope of the architecture and external dependencies described? |  |
| In the Dynamic design, is the functionality and behavior described? |  |
| In the Dynamic design, is the control flow and concurrency of processes described? |  |
| In the Dynamic design, is the data flow between the software components described? |  |
| In the Dynamic design, is the data flow at external interfaces described? |  |
| In the Dynamic design, is the temporal constraints described? |  |
| **7.4.6 REUSE** 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. | Are all safety related software component categorized as | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| a) newly developed; |  |
| b) reused with modifications; |  |
| c) reused without modifications. |  |
| **7.4.7 NEWLY** Safety-related software components that are newly developed or reused with modifications shall be developed in accordance with ISO 26262.  NOTE In these cases ISO 26262-8:2011, Clause 12, does not apply. | Are all safety related software component newly developed or reused with modifications developed in accordance with ISO 26262? | **WORK PRODUCT 7.5.2 Safety plan (refined)** resulting from requirement 7.4.7. |
| **7.4.8 REUSE** Safety-related software components that are reused without modifications shall be qualified in  accordance with ISO 26262-8:2011, Clause 12. | Are all safety related software component reused without modifications qualified in accordance with ISO 26262-8:2011, Clause 12? |  |
| **7.4.9 SSR🡨(ASIL)🡪SC**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. | Are The software safety requirements allocated to the software components? | **WORK PRODUCT 7.5.1 Software architectural design specification**  **WORK PRODUCT 7.5.3 Software safety requirements specification (refined)** |
| Are all software components developed in compliance with the highest ASIL of the requirements allocated to it? |  |
| **7.4.10 EMBEDDED SOFTWARE HIGHEST ASIL** 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. | Is the embedded software treated in accordance with the highest ASIL? | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| **7.4.11 SOFTWARE PARTITIONING (THREADS)** If software partitioning (see Annex D) is used to implement freedom from interference between software components it shall be ensured that:  a) **SHARED RESOURCES** 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) **SOFTWARE PARTITIONING** the software partitioning is supported by dedicated hardware features or equivalent means (this requirement applies to ASIL D, in accordance with 4.3);  c) **HIGHEST ASIL OF SW PARTITIONER** 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;  NOTE In general the operating system provides or supports software partitioning.  d) **VERIFICATION OF SW PARTITIONING** the verification of the software partitioning during software integration and testing (in accordance with Clause 10) is performed. | a) Are shared resources used in such a way that freedom from interference of software partitions is ensured? |  |
| b) is the software partitioning supported by dedicated hardware features or equivalent means (this requirement applies to ASIL D, in accordance with 4.3); |  |
| c) Is the part of the software that implements the software partitioning developed in compliance with the same or an ASIL higher than the highest ASIL assigned to the requirements of the software partitions? |  |
| d) Was the verification of the software partitioning during software integration and testing (in accordance with Clause 10) performed? |  |
|  |  |
| **7.4.12 DEPENDENT FAILURES** 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. |  | **WORK PRODUCT 7.5.5 Dependent failures analysis report** |
| If the implementation of software safety requirements relies on freedom from interference or sufficient independence between software components, was an analysis of dependent failures in accordance with ISO 26262-9:2011, Clause 7, **carried out**? |  |
| **7.4.13 SAFETY ANALYSIS** Safety analysis **shall be carried out** at the software architectural level in accordance with ISO 26262-9:2011, Clause 8, in order to:  a) identify or confirm the safety-related parts of the software;  b) 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. |  | **WORK PRODUCT 7.5.4 Safety analysis report** |
| Was a Safety analysis **carried out** at the software architectural level in accordance with ISO 26262-9:2011, Clause 8, in order: |  |
| a) To identify or confirm the safety-related parts of the software? |  |
| b) To support the specification and verify the efficiency of the safety mechanisms? |  |
| **7.4.14 SAFETY MECHANISM FOR ERROR DETECTION** 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 analyze the potential impact on the system behavior. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |

### Table 4 — Mechanisms for error detection at the software architectural level

<http://publications.lib.chalmers.se/records/fulltext/184469/184469.pdf>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Methods | | ASIL | | | | Status |
| A | B | C | D |
| 1a | Range checks of input and output data | ++ | ++ | ++ | ++ | Not applied |
| 1b | Plausibility check a | + | + | + | ++ | Not applied |
| 1c | Detection of data errors b | + | + | + | + | ECC mechanism |
| 1d | External monitoring facility c | o | + | + | ++ | Watchdog of the ECU |
| 1e | Control flow monitoring | o | + | ++ | ++ | Not applied |
| 1f | Diverse software design | o | o | + | ++ | Not applied |

a Plausibility checks can include using a reference model of the desired behavior, assertion checks, or comparing signals from different sources.

b Types of methods that may be used to detect data errors include error detecting codes and multiple data storage.

c An external monitoring facility can be, for example, an ASIC or another software element performing a watchdog function.

|  |  |  |
| --- | --- | --- |
| **7.4.15** This subclause applies to ASIL (A), (B), C and D, in accordance with 4.3: 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. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |

### Table 5 — Mechanisms for error handling at the software architectural level

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Methods | | ASIL | | | | Status |
| A | B | C | D |  |
| 1a | Static recovery mechanism | + | + | + | + | Watchdog mechanism |
| 1b | Graceful degradation | + | + | ++ | ++ | Safety monitor mechanism |
| 1c | Independent parallel redundancy | o | o | + | ++ | Not applied |
| 1d | Correcting codes for data | + | + | + | + | ECC mechanism |

|  |  |  |
| --- | --- | --- |
| **7.4.16 NEW HAZARDS** 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** 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. |  | **WORK PRODUCT 7.5.1 Software architectural design specification** |
| **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:  a) compliance with the software safety requirements;  b) compatibility with the target hardware; and  NOTE This includes the resources as specified in 7.4.17.  c) adherence to design guidelines. |  | **WORK PRODUCT 7.5.6 Software verification report (refined)** |

**Table 6 — Methods for the verification of the software architectural design**

**Methods ASIL A B C D**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Methods | | ASIL | | | | Status |
| A | B | C | D |
| 1a | Walk-through of the design a | ++ | + |  |  | Informal review |
| 1b | Inspection of the design a | + | ++ | ++ | ++ | Formal review |
| 1c | Simulation of dynamic parts of the design b | + | + | + | ++ | Not performed |
| 1d | Prototype generation |  |  | + | ++ | Not performed |
| 1e | Formal verification |  |  | + | + | Not applied |
| 1f | Control flow analysis c | + | + | ++ | ++ | Not performed |
| 1g | Data flow analysis c | + | + | ++ | ++ | Not performed |

a In the case of model-based development these methods can be applied to the model.

b Method 1c requires the usage of executable models for the dynamic parts of the software architecture.

c Control and data flow analysis may be limited to safety-related components and their interfaces.

**7.5 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.

# 8 Software unit design and implementation

|  |  |  |
| --- | --- | --- |
| **8.1 Objectives** |  |  |
| **Software Units** **Specification** The first objective of this sub-phase is to specify the software units in accordance with the software architectural design and the associated software safety requirements. |  |  |
| **Implement the Software Units** The second objective of this sub-phase is to implement the software units as specified. |  |  |
| **Static Verification of the Software units** The third objective of this sub-phase is the static verification of the design of the software units and their implementation. |  |  |

## 8.2 General

**SW IS DEVELOPED** Based on the software architectural design, the detailed design of the software units is developed. **MODEL OR SOURCE CODE** The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or coding guidelines respectively.

**STATIC VERIFICATION** The detailed design and the implementation are statically verified before proceeding to the software unit testing phase.

**MANUAL SOURCE CODE PROPERTIES** The implementation-related properties are achievable at the source code level if manual code development is used.

**AUTOMATIC SOURCE CODE PROPERTIES** If model-based development with automatic code generation is used, these properties apply to the model and need not apply to the source code.

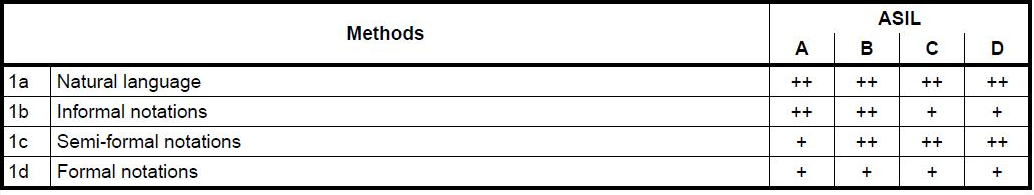
**SAFETY AND NON SAFETY RELATED REQUIREMENTS** In order to develop a single software unit design both software safety requirements as well as all non-safety related requirements are implemented. Hence in this sub-phase safety-related and non-safety-related requirements are handled within one development process.

**OBJECT CODE** The implementation of the software units includes the generation of source code and the translation into object code.

## 8.4 Requirements and recommendations

|  |  |  |
| --- | --- | --- |
| **8.4.1 ALL REQS COMPLIED** The requirements of this sub clause **shall be complied** with if the software unit is safety-related.  NOTE “Safety-related” means that the unit implements safety requirements, or that the criteria for coexistence (see  ISO 26262-9:2011, Clause 6) of the unit with other units are not satisfied. | Are all requirements of this sub clause complied? |  |
| **8.4.2** To ensure that the software unit design captures the information necessary to allow the subsequent development activities to be performed correctly and effectively, the software unit design **shall be described** using the notations listed in Table 7.  NOTE In the case of model-based development with automatic code generation, the methods for representing the notation software unit design are applied to the model which serves as the basis for the code generation. | What notation was used? | **WORK PRODUCT 8.5.1 Software unit design specification** |

### Table 7 — Notations for software unit design



Semi-formal notations: Autosar

|  |  |  |
| --- | --- | --- |
| **8.4.3 SW UNITS IMPLEMENTATION** The specification of the software units **shall describe** the functional behavior and the internal design to the level of detail necessary for their implementation.  EXAMPLE Internal design can include constraints on the use of registers and storage of data. | Does the specification describe the functional behavior and the internal design with necessary detail? | **WORK PRODUCT 8.5.1 Software unit design specification** |
| **8.4.4** Design principles for software unit design and implementation at the source code level as listed in Table 8 **shall be applied** to achieve the following properties: | Was the design principles listed in Table 8 applied? | **WORK PRODUCT 8.5.1 Software unit design specification and**  **WORK PRODUCT 8.5.2 Software unit implementation** |
| a) **ORDER OF EXECUTION** correct order of execution of subprograms and functions within the software units, based on the software architectural design; |  |  |
| b) **USE OF INTERFACES** consistency of the interfaces between the software units; |  |  |
| c) **DATA FLOW AND CONTROL FLOW** correctness of data flow and control flow between and within the software units; |  |  |
| d) **SIMPLICITY** simplicity; |  |  |
| e) **READABILITY AND COMPREHENSIBILITY** readability and comprehensibility; |  |  |
| f) **ROBUSTNESS** robustness; |  |  |
| EXAMPLE Methods to prevent implausible values, execution errors, division by zero, and errors in the data flow and control flow. |  |  |
| g) **MODIFICATION** suitability for software modification; and |  |  |
| h) **TASTABILITY** testability. |  |  |

#### Table 8 — Design principles for software unit design and implementation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Methods | | ASIL | | | |
| A | B | C | D |
| 1a | One entry and one exit point in subprograms and functions a (No Return in the middle of function) | ++ | ++ | ++ | ++ |
| 1b | No dynamic objects or variables, or else online test during their creation a,b (No Malloc) | + | ++ | ++ | ++ |
| 1c | Initialization of variables | ++ | ++ | ++ | ++ |
| 1d | No multiple use of variable names a | + | ++ | ++ | ++ |
| 1e | Avoid global variables or else justify their usage a | + | + | ++ | ++ |
| 1f | Limited use of pointers a | o | + | + | ++ |
| 1g | No implicit type conversions a,b | + | ++ | ++ | ++ |
| 1h | No hidden data flow or control flow c | + | ++ | ++ | ++ |
| 1i | No unconditional jumps a,b,c (Goto) | ++ | ++ | ++ | ++ |
| 1j | No recursions | + | + | ++ | ++ |

a Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

b Methods 1g and 1i are not applicable in assembler programming.

c Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

NOTE For the C language, MISRA C[3] covers many of the methods listed in Table 8.

|  |  |  |
| --- | --- | --- |
| **8.4.5 VERIFICATION** The software unit design and implementation **shall be verified** in accordance with ISO 26262-8:2011 Clause 9, and by applying the verification methods listed in Table 9, to demonstrate: | Was the software unit verified in accordance with ISO 26262-8:2011-9 and by applying table 9? | **WORK PRODUCT 8.5.3 Software verification report (refined)** |
| a) **HSI COMPLIANCE** the compliance with the hardware-software interface specification (in accordance with ISO 26262-5:2011, 6.4.10); | Was the HSI specification complied? |  |
| b) **TRACEABILITY** the fulfilment of the software safety requirements as allocated to the software units (in accordance with 7.4.9) through traceability; | Were all software safety requirements allocated to the software units? |  |
| Was traceability used? |  |
| c) **SOURCE CODE MATCH DESIGN** the compliance of the source code with its design specification;  NOTE In the case of model-based development, requirement c) still applies. |  |  |
| d) **SOURCE CODE MATCH GUIDELINES** the compliance of the source code with the coding guidelines (see 5.5.3); and |  |  |
| e) **COMPATIBILITY HW-SW** the compatibility of the software unit implementations with the target hardware. |  |  |

#### Table 9 — Methods for the verification of software unit design and implementation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Walk-through a | ++ | + | o | o |
| 1b | Inspection a | + | ++ | ++ | ++ |
| 1c | Semi-formal verification | + | + | ++ | ++ |
| 1d | Formal verification | o | o | + | + |
| 1e | Control flow analysis b,c | + | + | ++ | ++ |
| 1f | Data flow analysis b,c | + | + | ++ | ++ |
| 1g | Static code analysis | + | ++ | ++ | ++ |
| 1h | Semantic code analysis d | + | + | + | + |

a In the case of model-based software development the software unit specification design and implementation can be verified at the model level.

b Methods 1e and 1f can be applied at the source code level. These methods are applicable both to manual code development and to model-based development.

c Methods 1e and 1f can be part of methods 1d, 1g or 1h.

d Method 1h is used for mathematical analysis of source code by use of an abstract representation of possible values for the variables. For this it is not necessary to translate and execute the source code.

NOTE Table 9 lists only **static verification techniques**. Dynamic verification techniques (e.g. testing techniques) are

covered in Tables 10, 11 and 12.

# 9 Software unit testing

## 9.1 Objectives

**SW UNITS FULFILL SPECIFICATIONS** The objective of this sub-phase is to demonstrate that the software units fulfil the software unit design specifications and do not contain undesired functionality. See 8.4.2, 8.4.3 and 8.4.4

## 9.2 General

A procedure for testing the **software unit** against the **software unit design specifications** is established, and the tests are carried out in accordance with this procedure.

## 9.4 Requirements and recommendations

|  |  |
| --- | --- |
| **9.4.1 ALL REQS COMPLIED** The requirements of this sub clause **shall be complied** with if the software unit is safety-related.  NOTE “Safety-related” means that the unit implements safety requirements, or that the criteria for coexistence of the unit with other units are not satisfied. |  |
| **9.4.2 PLANNING OF UNIT TESTING** Software unit testing shall be planned, specified and executed in accordance with ISO 26262-8:2011, Clause 9.  NOTE 1 Based on the definitions in ISO 26262-8:2011, Clause 9, the test objects in the software unit testing are the software units.  NOTE 2 For model-based software development, the corresponding parts of the implementation model also represent objects for the test planning. Depending on the selected software development process the test objects can be the code derived from this model or the model itself. | **Work products** 9.5.1 Software verification plan (refined)  **9.5.2 Software verification specification**  **9.5.3 Software verification report (refined)** |
| **9.4.3** The software unit testing methods listed in Table 10 **shall be applied** to demonstrate that the software units achieve: | **Work products** 9.5.1 Software verification plan (refined) |
| a) compliance with the software unit design specification (in accordance with Clause 8); | Does SW Unit achive compliance with sw unit design specification through use of sw unit testing methods? |
| b) compliance with the specification of the hardware-software interface (in accordance with  ISO 26262-5:2011, 6.4.10); | Does SW Unit achive compliance with HSI through use of sw unit testing methods? |
| c) the specified functionality; | Does SW Unit achive compliance with specified functionality through use of sw unit testing methods? |
| d) confidence in the absence of unintended functionality; | Does SW Unit achive confidence in the absence of unintended functionality through use of sw unit testing methods? |
| e) robustness; and  EXAMPLE The absence of inaccessible software, the effectiveness of error detection and error handling  mechanisms. | Does SW Unit achive robustness through use of sw unit testing methods? |
| f) sufficient resources to support their functionality. | Does SW Unit achive sufficient resources to support their functionalities through use of sw unit testing methods? |
|  |  |

#### Table 10 — Methods for software unit testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Requirements-based test a | ++ | ++ | ++ | ++ |
| 1b | Interface test | ++ | ++ | ++ | ++ |
| 1c | Fault injection test b | + | + | + | ++ |
| 1d | Resource usage test c | + | + | + | ++ |
| 1e | Back-to-back comparison test between model and code, if applicable d | + | + | ++ | ++ |

a The software requirements at the unit level are the basis for this requirements-based test.

b This includes injection of arbitrary faults (e.g. by corrupting values of variables, by introducing code mutations, or by corrupting values of CPU registers).

c Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

d This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.

|  |  |
| --- | --- |
| **9.4.4** To enable the specification of appropriate test cases for the software unit testing in accordance with 9.4.3, test cases **shall be** derived using the methods listed in Table 11. | **Work products** 9.5.1 Software verification plan (refined)  **9.5.2 Software verification specification** |
| Was test cases derived from Table 11? |

#### Table 11 — Methods for deriving test cases for software unit testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Analysis of requirements | ++ | ++ | ++ | ++ |
| 1b | Generation and analysis of equivalence classes a | + | ++ | ++ | ++ |
| 1c | Analysis of boundary values b | + | ++ | ++ | ++ |
| 1d | Error guessing c | + | + | + | + |

a Equivalence classes can be identified based on the division of inputs and outputs, such that a representative test value can be selected for each class.

b This method applies to interfaces, values approaching and crossing the boundaries and out of range values.

c Error guessing tests can be based on data collected through a “lessons learned” process and expert judgment.

|  |  |
| --- | --- |
| **9.4.5 UNINTENDED FUNCTIONALITIES** To evaluate the completeness of test cases and to demonstrate that there is no unintended functionality, the coverage of requirements at the software unit level shall be determined and the **structural coverage** shall be measured in accordance with the metrics listed in Table 12.  If the achieved structural coverage is considered insufficient, either additional test cases shall be specified or a rationale shall be provided.  EXAMPLE 1 Analysis of structural coverage can reveal shortcomings in requirement-based test cases, inadequacies in requirements, dead code, deactivated code or unintended functionality.  EXAMPLE 2 A rationale can be given for the level of coverage achieved based on accepted dead code (e.g. code for debugging) or code segments depending on different software configurations; or code not covered can be verified using complementary methods (e.g. inspections). | **Work products** 9.5.1 Software verification plan (refined)  **9.5.2 Software verification specification** |

#### Table 12 — Structural coverage metrics at the software unit level

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Statement coverage | ++ | ++ | + | + |
| 1b | Branch coverage | + | ++ | ++ | ++ |
| 1c | MC/DC (Modified Condition/Decision Coverage) | + | + | + | ++ |

NOTE 1 The structural coverage can be determined by the use of appropriate software tools.

NOTE 2 In the case of model-based development, the analysis of structural coverage can be performed at the model

level using analogous structural coverage metrics for models.

NOTE 3 If instrumented code is used to determine the degree of coverage, it can be necessary to show that the

instrumentation has no effect on the test results. This can be done by repeating the tests with non-instrumented code.

|  |  |
| --- | --- |
| **9.4.6 TARGET ENVIRONMENT DIFFERENCES** The test environment for software unit testing shall correspond as closely as possible to the target environment.  If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases.  NOTE 1 Differences between the test environment and the target environment can arise in the source code or object code, for example, due to different bit widths of data words and address words of the processors.  NOTE 2 Depending on the scope of the tests, the appropriate test environment for the execution of the software unit is used (e.g. the target processor, a processor emulator or a development system).  NOTE 3 Software unit testing can be executed in different environments, for example:   * model-in-the-loop tests; * software-in-the-loop tests; * processor-in-the-loop tests; and * hardware-in-the-loop tests.   NOTE 4 For model-based development, software unit testing can be carried out at the model level followed by back-to-back comparison tests between the model and the object code. The back-to-back comparison tests are used to ensure that the behaviour of the models with regard to the test objectives is equivalent to the automatically-generated code. | **Work products** 9.5.1 Software verification plan (refined)  **9.5.2 Software verification specification** |

# 10 Software integration and testing

## 10.1 Objectives

**INTEGRATE SW ELEMENTS** The first objective of this sub-phase is to integrate the software elements.

**SW ARCHITECTURAL DESIGNED REALIZED BY EMBEDDED SW**  The second objective of this sub-phase is to demonstrate that the software architectural design is realized by the embedded software.

## 10.2 General

**INTERFACES TESTS AGAINST SW ARCHITECTURAL DESIGN** In this sub-phase, the particular integration levels and the interfaces between the software elements are tested against the software architectural design.

The steps of the integration and testing of the software elements correspond directly to the hierarchical architecture of the software.

The embedded software can consist of safety-related and non-safety-related software elements.

## 10.4 Requirements and recommendations

|  |  |  |
| --- | --- | --- |
|  |  | Evidence |
| **10.4.1 PLANNING** The planning of the software integration **shall describe** the steps for integrating the individual software units hierarchically into software components until the embedded software is fully integrated, and **shall consider**: | Are the steps for integrating the individual sw units hierarchicaly into sw components described in the planning? |  |
| **10.5.1 Software verification plan (refined)**  **10.5.2 Software verification specification (refined)**  **10.5.3 Embedded software** |  |
| a) the functional dependencies that are relevant for software integration; and | Are the functional dependencies that are relevant for software integration considered? |  |
| b) the dependencies between the software integration and the hardware-software integration. | Are the dependencies between sw integration and HSI considered? |  |
| NOTE For model-based development, the software integration can be replaced with integration at the model level and subsequent automatic code generation from the integrated model. |  |  |
| **10.4.2** Software integration testing **shall be planned**, specified and executed in accordance with ISO 26262-8:2011, Clause 9.  NOTE 1 Based on the definitions in ISO 26262-8:2011, Clause 9, the software integration test objects are the software components.  NOTE 2 For model-based development, the test objects can be the models associated with the software components. | Was the sw integration testing planned, specified ane excuted in accordance with ISO 26262-8:2011, Clause 9.  **10.5.1 Software verification plan (refined)**  **10.5.2 Software verification specification (refined)**  **10.5.4 Software verification report (refined)** |  |
| **10.4.3** The software integration test methods listed in Table 13 **shall be applied** to demonstrate that both the software components and the embedded software achieve: | **10.5.1 Software verification plan (refined)** |  |
| a) compliance with the software architectural design in accordance with Clause 7; | Was Table 13 applied to demonstrate that both the software components and the embedded software achieve compliance with the software architectural design in accordance with Clause 7? |  |
| b) compliance with the specification of the hardware-software interface in accordance with ISO 26262-4:2011, Clause 7; | Was Table 13 applied to demonstrate that both the software components and the embedded software achieve compliance with the specification of the hardware-software interface in accordance with ISO 26262-4:2011, Clause 7? |  |
| c) the specified functionality; | Was Table 13 applied to demonstrate that both the software components and the embedded software achieve the specified functionality? |  |
| d) robustness; and  EXAMPLE Absence of inaccessible software; effective error detection and handling. | Was Table 13 applied to demonstrate that both the software components and the embedded software achieve robustness? |  |
| e) sufficient resources to support the functionality. | Was Table 13 applied to demonstrate that both the software components and the embedded software achieve sufficient resources to support the functionality? |  |

#### Table 13 — Methods for software integration testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Requirements-based test a | ++ | ++ | ++ | ++ |
| 1b | Interface test | ++ | ++ | ++ | ++ |
| 1c | Fault injection test b | + | + | ++ | ++ |
| 1d | Resource usage test cd | + | + | + | ++ |
| 1e | Back-to-back comparison test between model and code, if applicable e | + | + | ++ | ++ |

a The software requirements at the architectural level are the basis for this requirements-based test.

b This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting software or hardware components).

c To ensure the fulfilment of requirements influenced by the hardware architectural design with sufficient tolerance, properties suchas average and maximum processor performance, minimum or maximum execution times, storage usage (e.g. RAM for stack and heap, ROM for program and data) and the bandwidth of communication links (e.g. data buses) have to be determined.

d Some aspects of the resource usage test can only be evaluated properly when the software integration tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

e This method requires a model that can simulate the functionality of the software components. Here, the model and code are stimulated in the same way and results compared with each other.

|  |  |  |
| --- | --- | --- |
| **10.4.4** To enable the specification of appropriate test cases for the software integration test methods  selected in accordance with 10.4.3, test cases **shall be derived** using the methods listed in Table 14. | Was test cases derived using methods listed in table 14?  **10.5.2 Software verification specification (refined)** |  |

#### Table 14 — Methods for deriving test cases for software integration testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Analysis of requirements | ++ | ++ | ++ | ++ |
| 1b | Generation and analysis of equivalence classes a | + | ++ | ++ | ++ |
| 1c | Analysis of boundary values b | + | ++ | ++ | ++ |
| 1d | Error guessing c | + | + | + | + |

a Equivalence classes can be identified based on the division of inputs and outputs, such that a representative test value can be

selected for each class.

b This method applies to parameters or variables, values approaching and crossing the boundaries and out of range values.

c Error guessing tests can be based on data collected through a “lessons learned” process and expert judgment.

|  |  |  |
| --- | --- | --- |
| **10.4.5 COMPLETENESS OF TESTS** To evaluate the completeness of tests and to obtain confidence that there is no unintended functionality, the coverage of requirements at the software architectural level by test cases **shall be determined.** If necessary, additional test cases shall be specified or a rationale shall be provided. | Was determined the coverage of requirements at the software architectural level by test cases?  **10.5.1 Software verification plan (refined)**  **10.5.2 Software verification specification (refined)** |  |
| **10.4.6 COMPLETENESS OF TEST CASES** This subclause applies to ASIL (A), (B), C and D, in accordance with 4.3: To evaluate the completeness of test cases and to obtain confidence that there is no unintended functionality, the structural coverage **shall be measured** in accordance with the metrics listed in Table 15. If the achieved structural coverage is considered insufficient, either additional test cases shall be specified or a rationale shall be provided.  EXAMPLE Analysis of structural coverage can reveal shortcomings (deficiências) in the requirement-based test cases, inadequacies in the requirements, dead code, deactivated code or unintended functionality. | Was measured the structural coverage in accordance with the metrics listed in Table 15?  **10.5.1 Software verification plan (refined)** |  |

#### Table 15 — Structural coverage metrics at the software architectural level

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Function coverage a | + | + | ++ | ++ |
| 1b | Call coverage b | + | + | ++ | ++ |

a Method 1a refers to the percentage of executed software functions. This evidence can be achieved by an appropriate software integration strategy.

b Method 1b refers to the percentage of executed software function calls.

NOTE 1 The structural coverage can be determined using appropriate software tools.

NOTE 2 In the case of model-based development, software architecture testing can be performed at the model level using analogous structural coverage metrics for models.

|  |  |  |
| --- | --- | --- |
| **10.4.7 It shall be verified** that the embedded software that is to be included as part of a production release in accordance with ISO 26262-4:2011, Clause 11, contains all the specified functions, and only contains other unspecified functions if these functions do not impair the compliance with the software safety requirements.  EXAMPLE In this context unspecified functions include code used for debugging or instrumentation.  NOTE If deactivation of these unspecified functions can be ensured, this is an acceptable means of compliance with this requirement. Otherwise the removal of such code is a change (see ISO 26262-8:2011, Clause 8). | Was it verified that the embedded software that is to be included as part of a production release in accordance with ISO 26262-4:2011, Clause 11, contains all the specified functions, and only contains other unspecified functions if these functions do not impair the compliance with the software safety requirements?  **10.5.2 Software verification specification (refined)** |  |
| **10.4.8 TARGET ENVIRONMENT** The test environment for software integration testing **shall correspond** as closely as possible to the target environment.  If the software integration testing is not carried out in the target environment, the differences in the source and object code and the differences between the test environment and the target environment shall be analysed in order to specify additional tests in the target environment during the subsequent test phases.  NOTE 1 Differences between the test environment and the target environment can arise in the source or object code, for example, due to different bit widths of data words and address words of the processors.  NOTE 2 Depending on the scope of the tests and the hierarchical level of integration, the appropriate test environments for the execution of the software elements are used. Such test environments can be the target processor for final integration, or a processor emulator or a development system for the previous integration steps.  NOTE 3 Software integration testing can be executed in different environments, for example:  -model-in-the-loop (MIL) tests;  -software-in-the-loop tests;  -processor-in-the-loop tests; and  -hardware-in-the-loop (HIL) tests. | Does the test environment correspond as closely as possible to the target environment?  **10.5.1 Software verification plan (refined)**  **10.5.2 Software verification specification (refined)** |  |

<https://www.add2.co.uk/applications/model-in-the-loop-testing-applications/>

# 11 Verification of software safety requirements

## 11.1 Objectives

**EMBEDDED SW FULFILLS SW REQS** The objective of this sub-phase is to demonstrate that the embedded software fulfils the software safety requirements.

## 11.2 General

The purpose of the verification of the software safety requirements is to demonstrate that the embedded software satisfies its requirements in the target environment.

|  |  |
| --- | --- |
| **11.4.1** The verification of the software safety requirements **shall be planned, specified and executed** in accordance with ISO 26262-8:2011, Clause 9. | Was the verification ofsw safety reqs planned, specified and axecuted?  **11.5.1 Software verification plan (refined)**  **11.5.2 Software verification specification (refined)**  **11.5.3 Software verification report (refined)** |
| **11.4.2** To verify that the embedded software fulfils the software safety requirements, tests **shall be conducted** in the test environments listed in Table 16.  NOTE Test cases that already exist, for example from software integration testing, can be re-used. | Were tests conducted in the tests environments listed in Table 16?  **11.5.1 Software verification plan (refined)**  **11.5.2 Software verification specification (refined)** |

#### Table 16 — Test environments for conducting the software safety requirements verification

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D |
| 1a | Hardware-in-the-loop | + | + | ++ | ++ |
| 1b | Electronic control unit network environmentsa | ++ | ++ | ++ | ++ |
| 1c | Vehicles | ++ | ++ | ++ | ++ |

a Examples include test benches partially or fully integrating the electrical systems of a vehicle, “lab-cars” or “mule” vehicles, and “rest of the bus” simulations.

|  |  |  |
| --- | --- | --- |
| **11.4.3** The testing of the implementation of the software safety requirements **shall be executed** on the target hardware. | Was the testing of the implementation of the software safety requirements executed in teh target hardware?  **11.5.1 Software verification plan (refined)**  **11.5.2 Software verification specification (refined)** |  |
| **11.4.4** The results of the verification of the software safety requirements **shall be evaluated** with regard to: | **11.5.3 Software verification report (refined)** |  |
| a) compliance with the expected results; | Were The results of the verification of the software safety requirements evaluated with regard to compliance with the expected results? |  |
| b) coverage of the software safety requirements; and | Were The results of the verification of the software safety requirements evaluated with regard to coverage of the software safety requirements? |  |
| c) pass or fail criteria. | Were The results of the verification of the software safety requirements evaluated with regard to pass or fail criteria? |  |

# Annex D (informative)

## Freedom from interference between software elements

### D.1 Objectives

The objective is to provide examples of faults that can cause interference between software elements (e.g. software elements of different software partitions).

Additionally, this Annex provides examples of possible mechanisms that can be considered for the prevention, or detection and mitigation of the listed faults.

NOTE The capability and effectiveness of the mechanisms used to prevent, or to detect and mitigate relevant faults is assessed during development.

### D.2 General

#### D.2.1 Achievement of freedom from interference

To develop or evaluate the achievement of freedom from interference between software elements, the effects of the exemplary faults, and the propagation of the possible resulting failures can be considered.

#### D.2.2 Timing and execution

With respect to timing constraints, the effects of faults such as those listed below can be considered for the software elements executed in each software partition:

1. blocking of execution;
2. deadlocks;
3. livelocks;
4. incorrect allocation of execution time;
5. incorrect synchronization between software elements.

EXAMPLE Mechanisms such as

* cyclic execution scheduling,
* fixed priority based scheduling,
* time triggered scheduling,
* monitoring of processor execution time,
* program sequence monitoring and
* arrival rate monitoring can be considered.

#### D.2.3 Memory

With respect to memory, the effects of faults such as those listed below can be considered for software elements executed in each software partition:

1. corruption of content;
2. read or write access to memory allocated to another software element.

EXAMPLE Mechanisms such as

* memory protection,
* parity bits,
* error-correcting code (ECC),
* cyclic redundancy check (CRC),
* redundant storage,
* restricted access to memory,
* static analysis of memory accessing software and
* static allocation can be used.

#### D.2.4 Exchange of information

With respect to the exchange of information, the causes for faults or effects of faults such as those listed below can be considered for each sender or each receiver:

1. repetition of information;
2. loss of information;
3. delay of information;
4. insertion of information;
5. masquerade or incorrect addressing of information;
6. incorrect sequence of information;
7. corruption of information;
8. asymmetric information sent from a sender to multiple receivers;
9. information from a sender received by only a subset of the receivers;
10. blocking access to a communication channel.

NOTE The exchange of information between elements executed in different software partitions or different ECUs

includes signals, data, messages, etc.

EXAMPLE 1 Information can be exchanged using I/O-devices, data busses, etc.

EXAMPLE 2 Mechanisms such as

1. communication protocols,
2. information repetition,
3. loop back of information,
4. acknowledgement of information,
5. appropriate configuration of I/O pins,
6. separated point-to-point unidirectional communication objects,
7. unambiguous bidirectional communication objects,
8. asynchronous data communication,
9. synchronous data communication,
10. event-triggered data buses,
11. event-triggered data buses with time-triggered access,
12. time-triggered data busses,
13. mini-slotting,
14. bus arbitration by priority can be used.

EXAMPLE 3 Communication protocols can contain information such as

1. identifiers for communication objects,
2. keep alive messages,
3. alive counters,
4. sequence numbers,
5. error detection codes and
6. error-correcting codes.