Software V&V stategy (SWVVS)

|  |  |  |
| --- | --- | --- |
| **OP’nSoft Project information** | | |
| **Project ID** | **Project Name** | **Project Manager** |
| [Type Project ID] | [Type Project Name] | [Type Project Manager] |
| **Field of application:** [Type field of application] | | |
| **Customer Name** | **Project Start Date** | **OP’nSoft project ID** |
| [Type Customer Name ] | [Type Project Start Date ] | [Type OP’nSoft project ID] |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name | Function | Signature |
| Edited by |  |  |  |
| Reviewed by |  |  |  |
| Reviewed by |  |  |  |
| Reviewed by |  |  |  |
| Approved by |  |  |  |

# Template Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Author | Section | Description / Task ID |
| 10/01/2024 | 1.0 | François Glénard | All | Creation of the template for OP’nSoft |
|  |  |  |  |  |

# Document Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date  **(yyyy-dd-mm)** | Version  (x.y) | Status | Author | Section | Description / Task ID |
|  |  | Choose an item. |  |  |  |
|  |  | Choose an item. |  |  |  |
|  |  | Choose an item. |  |  |  |
|  |  | Choose an item. |  |  |  |

# Table of Contents

[Template Revision History 2](#_Toc155788758)

[Document Revision History 2](#_Toc155788759)

[Table of Contents 3](#_Toc155788760)

[1 Introduction 6](#_Toc155788761)

[1.1 Purpose and Scope 6](#_Toc155788762)

[1.2 Review and Approval 6](#_Toc155788763)

[1.3 Referenced documents 6](#_Toc155788764)

[1.4 Applicable documents 7](#_Toc155788765)

[1.5 Abbreviations 7](#_Toc155788766)

[2 Test approaches 8](#_Toc155788767)

[2.1 Handling of internal development objects 8](#_Toc155788768)

[2.2 Handling of third-party development objects 8](#_Toc155788769)

[2.3 Test Prioritization 8](#_Toc155788770)

[2.4 Variant Management 8](#_Toc155788771)

[2.5 Functional Safety based components and features (If Applicable) 8](#_Toc155788772)

[3 Software unit test strategy 10](#_Toc155788773)

[3.1 Unit Test objectives 10](#_Toc155788774)

[3.2 Pre-conditions for unit testing 10](#_Toc155788775)

[3.3 Unit Test scope 10](#_Toc155788776)

[3.4 Unit Test environment and tools 11](#_Toc155788777)

[3.4.1 Environment 11](#_Toc155788778)

[3.4.2 Tools 11](#_Toc155788779)

[3.5 Static unit test 11](#_Toc155788780)

[3.6 Dynamic Unit Test Design Strategy 11](#_Toc155788781)

[3.6.1 Test case design method activities 12](#_Toc155788782)

[3.6.2 Test case execution method categories 12](#_Toc155788783)

[3.6.3 Test cases content 13](#_Toc155788784)

[3.6.4 Test case priority 13](#_Toc155788785)

[3.6.5 Test criteria 13](#_Toc155788786)

[3.7 Test case/step result defect severity 14](#_Toc155788787)

[3.8 Regression Test Strategy 14](#_Toc155788788)

[4 Software Integration strategy 15](#_Toc155788789)

[4.1 Pre-conditions for integration 15](#_Toc155788790)

[4.2 Software integration environment and tools 15](#_Toc155788791)

[4.2.1 Integration environment 15](#_Toc155788792)

[4.2.2 Integration tools 15](#_Toc155788793)

[4.3 Software Integration process 15](#_Toc155788794)

[4.3.1 Integration Methods 15](#_Toc155788795)

[4.3.2 Integration scope and sequence 16](#_Toc155788796)

[5 Software Integration test strategy 16](#_Toc155788797)

[5.1 Integration test objective 16](#_Toc155788798)

[5.2 Pre-conditions for software integration testing 16](#_Toc155788799)

[5.3 Integration test scope 16](#_Toc155788800)

[5.4 Software integration test environment and tools 17](#_Toc155788801)

[5.4.1 Integration test environment 17](#_Toc155788802)

[5.4.2 Integration test tools 17](#_Toc155788803)

[5.5 Integration Test Design Strategy 17](#_Toc155788804)

[5.5.1 Integration test case design method activities 17](#_Toc155788805)

[5.5.2 Integration test case execution method categories 18](#_Toc155788806)

[5.5.3 Integration test cases content 19](#_Toc155788807)

[5.5.4 Integration test case priority 19](#_Toc155788808)

[5.5.5 Integration test criteria 20](#_Toc155788809)

[5.6 Integration test case/step result defect severity 20](#_Toc155788810)

[5.7 Integration testing regression test strategy 20](#_Toc155788811)

[6 Qualification test strategy 22](#_Toc155788812)

[6.1 Qualification test objective 22](#_Toc155788813)

[6.2 Pre-conditions for qualification testing 22](#_Toc155788814)

[6.3 Qualification test scope 22](#_Toc155788815)

[6.4 Qualification test environment and tools 23](#_Toc155788816)

[6.4.1 Qualification Test Environment 23](#_Toc155788817)

[6.4.2 Qualification Test Tools 23](#_Toc155788818)

[6.5 Qualification Test Design Strategy 23](#_Toc155788819)

[6.5.1 Qualification test case design method activities 23](#_Toc155788820)

[6.5.2 Qualification test case execution method categories 24](#_Toc155788821)

[6.5.3 Qualification test cases content 25](#_Toc155788822)

[6.5.4 Qualification test case priority 25](#_Toc155788823)

[6.5.5 Qualification test criteria 26](#_Toc155788824)

[6.6 Qualification test case/step result defect severity 26](#_Toc155788825)

[6.7 Qualification testing Regression Test Strategy 26](#_Toc155788826)

[7 Process tailoring 27](#_Toc155788827)

# Introduction

## Purpose and Scope

**Purpose:** The purpose of this Software V&V strategy is to define the overall approach that will be taken when delivering testing services to the [Type Project Name] Project. The Software V&V strategy document is created during the planning phase of the project.

This Software V&V strategy shall define amongst others:

* The test strategy
  + Prioritization of tests
  + Re-test and regression test strategy
* Test design
  + Test methods to be applied to this project

**Scope:** This Software V&V strategy is valid for the [Type Project Name] Project , it affects the software test activities and is part of the OP’nSoft test processes.

The Test V&V strategy is valid for the whole software development life cycle.

## Review and Approval

The Software V&V Strategy is intended for the project team and should be modified/adapted according to the project organization needs.

After the initial creation or any update afterwards, the document needs to be approved by all the following parties below:

* Software Proxy Product Owner
* Software Product Owner
* Software Quality Engineer

## Referenced documents

<In the table below, list the all the technical documents, norms, standards, etc, that were used to define this document>

|  |  |  |  |
| --- | --- | --- | --- |
| **Document name** | **Version** | **Date** | **Storage path and link** |
| Automotive SPICE - Process Reference and Assessment Model | 3.1 | 01/11/2017 | [Link](https://confluence.engine.capgemini.com/x/oFsoAQ) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Applicable documents

<In the table below, list the all the project specific documents that are cited in this document or that were used to define this document>

|  |  |  |  |
| --- | --- | --- | --- |
| **Document name** | **Version** | **Date** | **Storage path and link** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Abbreviations

<In the table below, list and describe the abbreviations that are used in the document>

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| ASIL | Automotive Safety Integrity Levels |
| FuSa | Functional Safety |
| HIL | Hardware In the Loop |
| MIL | Model In the Loop |
| SIL | Software In the Loop |
| V&V | Verification and Validation |

# Test approaches

## Handling of internal development objects

All internally developed objects shall be tested according to this test plan

## Handling of third-party development objects

The integration of third-party software like drivers shall be tested implicitly with integration and system tests. SW unit tests is not applicable.

## Test Prioritization

The priority of specifying test cases, test case implementation and test execution will be decided by the Software Validation Engineer and Software Product Owner.

The following points shall be considered while deciding priority.

* 1. Requirements are assigned to specific releases, based on this assignment test cases are created & tested to the respective releases.
  2. Tests related to safety requirements shall be treated with the highest priority.
  3. Tests related to functional requirements are more important than test related to detailed designs or code metrics.
  4. Priority of re-tests: severity/priority attribute of tasks/defects may indicate which tests to be adjusted and executed first.
  5. New tests may be more important than re-tests.

## Variant Management

Variants are listed below for [Type Project Name]

<List of Variant>

## Functional Safety based components and features (If Applicable)

**White box Testing:**

<Update>

**Black Box Testing:**

<Update>

# Software unit test strategy

## Unit Test objectives

|  |  |  |  |
| --- | --- | --- | --- |
| **Linked process** | **Objective** | **Measurable target** | **Responsibility** |
| SWE.4 Software Unit Verification | Coverage of MISRA | Recommended: 100% of manual code checked for MISRA compliance |  |
| SWE.4 Software Unit Verification | Cyclomatic complexity of SW Units managed | Recommended: <15 |  |
| SWE.4 Software Unit Verification | McDc : Modified condition Decision coverage | Recommended: 100% |  |
| SWE.4 Software Unit Verification | Cyclomatic complexity of Auto Code Generated from MATLAB | Recommended: <15 (TBC through MATLAB configuration) |  |
| SWE.4 Software Unit Verification | Coverage of Software Detailed Design units | 100% of SW unit Test Cases Linked back to Software Detailed Design |  |

## Pre-conditions for unit testing

Before starting unit testing, the following condition have to be met:

1. The software detailed design of the software component is baselined.

## Unit Test scope

The objective of the test is to find the bugs in software unit function level and to check that functionality of module as outlined in the detailed design (including verification of boundary values and robustness) cover 100% of the software detailed design.

<Fill the following table with the main tested objects, their key test points, test mode. Table is pre-filled as an example, make sure to tailor it according to the project>

|  |  |  |
| --- | --- | --- |
| **Tested aspect** | **Key Test Points** | **Test mode** |
| Static | MISRA | Manual |
| Cyclomatic complexity | Manual |
| Dynamic | Unit Function test cases | Manual |
| Mcdc | Manual |
|  |  |
|  |  |

## Unit Test environment and tools

### Environment

<Describe/illustrate the test environment that will be used for unit testing>

### Tools

<Fill the table below with information about the tools that will be used for unit testing>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Device** | **Internally produced / Manufacturer** | **Specification / Property / Version** | **Description of the tool application** | **Availability** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Static unit test

**Test method: Code Review**

Test focus: functionality, robustness, portability

Test object: source code

Test completion criteria: no non-accepted findings

Test tool: <Update> (recommended: GIT)

Responsible: Software Development Engineer

**Test method: Code Metric Analysis**

Test focus: low complexity, comment density

Test basis: software requirements

Test object: source code

Test completion criteria: below defined metric thresholds (MISRA + Cyclomatic complexity)

Test tool: <Update> (recommended : Coverity)

Responsible: Software Development Engineer

## Dynamic Unit Test Design Strategy

The purpose of the software unit test is to ensure that the software is tested to provide evidence for compliance with the softwrware requirements including functional safety requirements.

|  |  |
| --- | --- |
| **Test level** | Software Unit test |
| **Test object** | Software Units |
| **Test basis** | Software Detailed Design |
| **Test Objective** | Carry out a series of tests on the software units to find bugs and ensure compliance with software Detailed Design |
| **Test method context** | White box testing |

### Test case design method activities

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Design Method activities** | **Description** | **ASIL** | | | |
| **A** | **B** | **C** | **D** |
| Analysis of boundary values | The boundary value analysis method is a white box test method for testing the input or output boundary value. Check whether the upper, lower, and off points of the data can be processed correctly. |  |  | x | x |
| Analysis of requirements | Analyze the software requirements (including cybersecurity requirements, cybersecurity controls requirements, Functional safety requirements or other specific requirements) to develop relevant test case. | x | x | x | x |
| Error guessing based on knowledge or experience | During program testing, individuals can use their experience and knowledge to anticipate potential errors in the program and develop specific test methods to check for these errors. |  |  | x | x |

### Test case execution method categories

|  |  |  |
| --- | --- | --- |
| **Execution Method** | **Description** | **Relevant for the project** |
| Requirement-based test | Software requirements and specific test requirements are used for testing. This includes testing related to cybersecurity and functional safety requirements. When it comes to cybersecurity requirements, test cases are designed based on attack patterns or STRIDE threat modeling. Additionally, cybersecurity test cases should also be created to verify the effectiveness of cybersecurity controls or measures | <Yes/no + justification if no> |
| Fuzz Test | Fuzz testing, or fuzzing, is a method in software testing where you intentionally feed incorrect or random data (called "fuzz") into a software system. The goal is to **find coding errors and security vulnerabilities**. In fuzz testing, the emphasis is primarily on the data that comes from external sources or interfaces of the software. You provide a wide range of valid and invalid input data, and you also randomize the timing of signals. Then, you check how the software behaves under these different conditions to identify potential issues. | <Yes/no + justification if no> |
| Penetration Test | Penetration testing assesses the security of an application to uncover vulnerabilities. It's crucial to confirm the **software's resistance to cyberattacks** before system-level testing. This involves verifying essential cybersecurity functions like key management, secure configuration, secure boot, software updates, secure communication channels, memory access rights, diagnostic services, and protection against brute force attacks. | <Yes/no + justification if no> |
| Load test | Software running load, communication load and other related tests | <Yes/no + justification if no> |

### Test cases content

The test cases specification must contain the following information:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **ID** | Unique test case ID |
| **Title** | Title of the test case |
| **Status** | Indicate if the test case is Active / Not Active |
| **Level** | **Unit test level** |
| **Purpose** | Describe the purpose of the test case |
| **Priority** | Indicate if the test case if of High, Medium, or Low priority for it’s execution |
| **Environment** | Indicate the test environment for the test case |
| **Method** | Describe the method that will be used for conducting the test case (see section 3.6.2) |
| **Pre-condition** | Define the condition(s) that have to be met to start the test |
| **Component and unit** | Indicate the component and unit to which the test case belongs |
| **FuSa** | Indicate if the test case is Functional Safety relevant |
| **Cybersecurity** | Indicate if the test case is Cybersecurity relevant |
| **Steps** | Create as many steps as required |
| **Steps titles** | Title of each step of the test case |
| **Steps descriptions** | Describe each step of the test case |
| **Steps expected results** | Describe the expected result of each step of the test case |

### Test case priority

|  |  |
| --- | --- |
| **Test priority** | **Description** |
| **High** | Basic functions of the software, that affects the entire software function or cybersecurity and functional safety function |
| **Medium** | Basic functions of the module, tests that affect the module function. |
| **Low** | Small area of influence, could continue to perform tests on other functional modules |

### Test criteria

|  |  |
| --- | --- |
| **Test completion criteria** | There are a set of releases in the project. For each release a specific set of software units are delivered. The test completion applies the test objectives in chapter 3.1 only for those elements that need to be covered for this specific release. |
| **Test start criteria** | Static code analysis shall be completed without any error and warnings, and source code should compile |
| **Test end criteria** | All test objectives for software unit testing are achieved according to the chapter 3.1 test objectives. In addition, all defects should be closed from developer’s side, all expected and actual results are captured and documented with the test script, and 100% SWDD (Software Detailed Design) shall link to test cases. |
| **Test abort criteria** | if any stopper bug/blocker arises, testing shall be aborted |
| **Test restart criteria** | if any stopper bug/blocker arises, testing shall be aborted and after fixing bug, testing shall be restarted |

## Test case/step result defect severity

For detail about software defect tracking, please refer to the [SUP.9 – Problem Resolution Management Process](https://confluence.engine.capgemini.com/x/vKtVAQ).

|  |  |
| --- | --- |
| **Defect severity** | **Description** |
| **Blocker** | Critical defects causing system or application crashes, freezes, system hangs, or loss of data, complete loss of major functions, cybersecurity function or functional safety abnormal |
| **High** | A major defect refers to the failure to implement a function or feature, the loss of a major function, causing a serious problem. |
| **Normal** | Less serious errors. Although such defects do not affect the basic use of the system, but if not better. |
| **Minor** | Some minor problems have little effect on the function, and the product and attributes can still be used, such as individual typos and irregularly arranged text. |

## Unit Testing Regression Test Strategy

Regression test shall verify that implemented SW changes have no unintentional "side effects" on software function that have not been changed.

A regression test shall be performed on basic functionality and safety relevant implementations.

Test cases, relevant for regression test, shall be marked in the test specification.

During regression testing with unchanged test cases shall prove as per below points.

1) Unchanged test cases running on unchanged code shall "PASS"

2) Unchanged test cases running on changed code shall "FAIL"

3) Changed test cases on changed code shall "PASS"

The regression testing shall include re-running of previously completed tests and to verify fixture of current reported faults and re-emergence of previously fixed faults. Regression test shall be always performed when any changes done in software detailed design for respective modules in every milestone of project.

Test automation coverage: <Update>

Test Execution: Every Minor/Major release

Test automation time: < Update as per test automation coverage >.

# Software Integration strategy

## Pre-conditions for integration

Before starting software integration, the following conditions have to be met:

1. The Software Architectural Design document, including cybersecurity and functional safety content, is baselined.
2. The software unit tests are passed, and test results are baselined.

## Software integration environment and tools

### Integration environment

<Describe/illustrate the test environment that will be used for software integration>

### Integration tools

<Fill the table below with information about the tools that will be used for software integration>

|  |  |  |
| --- | --- | --- |
| **Tool** | **Version** | **Purpose** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Software Integration process

### Integration Methods

The table below is a list of general software integration methods. Different methods can be used for different parts of the software:

|  |  |
| --- | --- |
| **Method** | **Description** |
| Big Bang | All components or modules are integrated at once and are then tested as a unit. |
| Bottom up | Starting from the lowest-level components with the least dependencies, the integration is carried out layer by layer according to the architectural design order. |
| Top down | Starting from the top-level control module component, the design sequence execution is adopted for testing. |

<Indicate which method(s) has been chosen for the project and justify why>

### Integration scope and sequence

<Describe the integration scope and the integration sequence for the project>

# Software Integration test strategy

## Integration test objective

|  |  |  |  |
| --- | --- | --- | --- |
| **Linked process** | **Objective** | **Measurable target** | **Responsibility** |
| SWE.5  Software Integration and Integration Test | Software Architecture Design Component Interface Coverage | 100% |  |
| Selected functional Test Case Pass Rate (Including Bug Repair Test Item） | 100% |  |
| Component interfaces boundary test | 100% |  |

## Pre-conditions for software integration testing

Before starting software integration testing, the following conditions have to be met:

1. Software integration activities are completed.

## Integration test scope

The mainly test key point of software integration test is the interface between software components, resource consumption, dynamic behavior. Test modes include manual test, automatic test and calculate resource consumption.

<Fill the following table with the main tested objects, their key test points, test mode. Table is pre-filled as an example, make sure to tailor it according to the project>

|  |  |  |
| --- | --- | --- |
| **Object** | **Key Test Points** | **Test mode** |
| Interfaces of software components | Test the consistency of output interfaces value and input interfaces value in the full range and set the amplitude of corresponding output signals according to real requests. | Manual test |
| Resource consumption | The use of all components and total RAM, ROM, DFLASH, EEPROM, NVRAM use (.map) | Calculate resource consumption |
| The use of all components and total STACK use | Manual test |
| The use of all components and total CPU load | Manual test |
| Dynamic behavior | Trigger period and runnable | Manual test |
| Start up and shutdown process,  Sleep and wakeup process | Manual test |
| Component features test cases | Manual test |

## Software integration test environment and tools

### Integration test environment

<Describe/illustrate the test environment that will be used for integration testing>

### Integration test tools

<Fill the table below with information about the tools that will be used for integration testing>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Device** | **Internally produced / Manufacturer** | **Specification / Property / Version** | **Description of the tool application** | **Availability** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Integration Test Design Strategy

The purpose of the software integration tests is to verify the interactions between software components, resource consumption and dynamic behavior.

|  |  |
| --- | --- |
| **Test level** | Software Integration test |
| **Test object** | Software Components and their interfaces |
| **Test basis** | Software Architectural Design and Software Detailed Design |
| **Test Objective** | Verify the validity of the software architectural design by conducting software integration tests on software components interfaces, software resource consumption and software dynamic behavior, while ensuring the accuracy of software Integration. |
| **Test method context** | White box testing |

### Integration test case design method activities

The integration test case design methods can be a combination of the methods described in the following table. Test case design related to functional safety also need to comply with the ISO 26262-6:2018(E): Table 10 — Methods for verification of software integration.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Design Method activities** | **Description** | **ASIL** | | | | **Cybersecurity** |
| **A** | **B** | **C** | **D** |
| Analysis of boundary values | The boundary value analysis method is a white box test method for testing the input or output boundary value. Check whether the upper, lower, and off points of the data can be processed correctly. |  |  | x | x |  |
| Analysis of requirements | Analyze the software requirements (including cybersecurity requirements, cybersecurity controls requirements, Functional safety requirements or other specific requirements) to develop relevant test case. | x | x | x | x |  |
| Equivalence class generation and analysis | Divide a large dataset into different ranges where the testing effects are similar, making each range a representative group. Then, pick specific data points from each range to reflect the test results within that range. |  | x | x | x |  |
| Error guessing based on knowledge or experience | During program testing, individuals can use their experience and knowledge to anticipate potential errors in the program and develop specific test methods to check for these errors. |  |  | x | x |  |
| Causality graph method | Use a graphical method to illustrate different input combinations and then create a decision table to design the corresponding test cases. |  |  |  |  |  |
| STRIDE threat modelling analysis | Derive the test case from STRIDE threat modelling analysis, which analyze the cybersecurity properties and potential compromise method. The compromise points out the way to design appropriate cybersecurity test cases. |  |  |  |  | x |

### Integration test case execution method categories

|  |  |  |
| --- | --- | --- |
| **Execution Method** | **Description** | **Relevant for the project** |
| Requirement-based test | Software requirements and specific test requirements are used for testing. This includes testing related to cybersecurity and functional safety requirements. When it comes to cybersecurity requirements, test cases are designed based on attack patterns or STRIDE threat modeling. Additionally, cybersecurity test cases should also be created to verify the effectiveness of cybersecurity controls or measures | <Yes/no + justification if no> |
| Interface test | 1.For the interfaces after software integrated, conduct data input and output test, boundary value test, equivalent test, etc. And conduct the complete test on specific interfaces, compatibility, timing, and other specific levels of the software. Verification of the interface between component by testing the dynamic behavior of the software component;  2.Test the interface by fuzzy data. The data transmitted over interface should be enumerated and random value is given. Different data values are combined to generate different test case combination. Random timing is another factor to generate the test case. | <Yes/no + justification if no> |
| Fuzz Test | Fuzz testing, or fuzzing, is a method in software testing where you intentionally feed incorrect or random data (called "fuzz") into a software system. The goal is to **find coding errors and security vulnerabilities**. In fuzz testing, the emphasis is primarily on the data that comes from external sources or interfaces of the software. You provide a wide range of valid and invalid input data, and you also randomize the timing of signals. Then, you check how the software behaves under these different conditions to identify potential issues. | <Yes/no + justification if no> |
| Penetration Test | Penetration testing assesses the security of an application to uncover vulnerabilities. It's crucial to confirm the **software's resistance to cyberattacks** before system-level testing. This involves verifying essential cybersecurity functions like key management, secure configuration, secure boot, software updates, secure communication channels, memory access rights, diagnostic services, and protection against brute force attacks. | <Yes/no + justification if no> |
| Fault injection test | Tests related to software injection into hardware or other interface faults (including the functional safety fault） | <Yes/no + justification if no> |
| Load test | Software running load, communication load and other related tests | <Yes/no + justification if no> |
| Trust Boundary Test | Trust Boundary Test is verifying dataflow or operation is secured when it needs to cross the different trust zone or security level in different modules. | <Yes/no + justification if no> |

### Integration test cases content

The test cases specification must contain the following information:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **ID** | Unique test case ID |
| **Title** | Title of the test case |
| **Status** | Indicate if the test case is Active / Not Active |
| **Level** | **Integration test level** |
| **Purpose** | Describe the purpose of the test case |
| **Priority** | Indicate if the test case if of High, Medium, or Low priority for it’s execution |
| **Environment** | Indicate the test environment for the test case |
| **Method** | Describe the method that will be used for conducting the test case (see section 5.5.2) |
| **Pre-condition** | Define the condition(s) that have to be met to start the test |
| **Functional module** | Indicate the functional module to which the test case belongs |
| **FuSa** | Indicate if the test case is Functional Safety relevant |
| **Cybersecurity** | Indicate if the test case is Cybersecurity relevant |
| **Steps** | Create as many steps as required |
| **Steps titles** | Title of each step of the test case |
| **Steps descriptions** | Describe each step of the test case |
| **Steps expected results** | Describe the expected result of each step of the test case |

### Integration test case priority

|  |  |
| --- | --- |
| **Test priority** | **Description** |
| **High** | Basic functions of the software, that affects the entire software function or cybersecurity and functional safety function |
| **Medium** | Basic functions of the module, tests that affect the module function. |
| **Low** | Small area of influence, could continue to perform tests on other functional modules |

Note: Basic functionality is the functionality that affects the overall software. if there is a problem with this functionality, the testing of the entire software functionalities will not be possible.

### Integration test criteria

|  |  |
| --- | --- |
| **Test completion criteria** | There are a set of releases in the project. For each release a specific set of software architecture elements are delivered. The test completion applies the test objectives in section 5.1 only for those elements that need to be covered for this specific release. |
| **Test start criteria** | Integrated software shall be available without any error and warnings, and source code should compile |
| **Test end criteria** | All test objectives for software integration testing are achieved according to the chapter 5.1 test objectives. In additional bugs should be closed and all expected and actual results are captured and documented with the test script, 100% Software Architectural Design shall link to test cases |
| **Test abort criteria** | if any stopper bug/blocker arises, testing shall be aborted |
| **Test restart criteria** | if any stopper bug/blocker arises, testing shall be aborted and after fixing bug, testing shall be restarted |

## Integration test case/step result defect severity

For detail about software defect tracking, please refer to the [SUP.9 – Problem Resolution Management Process](https://confluence.engine.capgemini.com/x/vKtVAQ).

|  |  |
| --- | --- |
| **Defect severity** | **Description** |
| **Blocker** | Critical defects causing system or application crashes, freezes, system hangs, or loss of data, complete loss of major functions, cybersecurity function or functional safety abnormal |
| **High** | A major defect refers to the failure to implement a function or feature, the loss of a major function, causing a serious problem. |
| **Normal** | Less serious errors. Although such defects do not affect the basic use of the system, but if not better. |
| **Minor** | Some minor problems have little effect on the function, and the product and attributes can still be used, such as individual typos and irregularly arranged text. |

## Integration testing regression test strategy

Regression test shall verify that implemented SW changes have no unintentional "side effects" on software function that have not been changed.

A regression test shall be performed on basic functionality and safety relevant implementations.

During regression testing with unchanged test cases shall prove as per below points.

1) Unchanged test cases running on unchanged code shall "PASS"

2) Unchanged test cases running on changed code shall "FAIL"

3) Changed test cases on changed code shall "PASS"

The Regression testing shall include re-running of previously completed tests and to verify fixture of current reported faults and re-emergence of previously fixed faults. Regression test shall be always performed when any changes done in High level design for respective modules in every milestone of project.

Tests scope: For that design component where requirements changed all test cases design components are retested.

Test automation coverage: <Update>

Test Execution: Every Minor/Major release

Test automation time: < Update as per test automation coverage >.

# Qualification test strategy

## Qualification test objective

|  |  |  |  |
| --- | --- | --- | --- |
| **Linked process** | **Objective** | **Measurable target** | **Responsibility** |
| SWE.6  Software Qualification Test | Coverage of all software requirements and functionality | 100% of software requirements shall link to test cases |  |

## Pre-conditions for qualification testing

Before starting qualification testing, the following conditions have to be met:

1. The software requirement specification is baselined.
2. The software integration tests are passed, and test results are baselined.

## Qualification test scope

The software qualification test is mainly a black box test. The test object is regarded as a black box. Regardless of the internal logical structure of the software, only the requirements specifications are used to check whether the function of the test object meets its functional description.

<Fill the following table with the main functions of the software, their key test points, test environment and test mode. The first 2 lines show an example that you should remove when filling the template>

|  |  |  |  |
| --- | --- | --- | --- |
| **Software Main Functions** | **Key Test Points** | **Test environment** | **Test mode** |
| Battery Data Collection | Cell Voltage Sampling: Validity of collected data (including the validity of the first frame of data, the sending cycle, and the sampling range) | HIL | Manual test |
| Model Temperature Sampling: Validity of collected data (including the validity of the first frame of data, the sending cycle, and the sampling range) | HIL | Manual test |
|  |  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Qualification test environment and tools

### Qualification Test Environment

<Describe/illustrate the test environment that will be used for qualification testing>

### Qualification Test Tools

<Fill the table below with information about the tools that will be used for qualification testing>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Device** | **Internally produced / Manufacturer** | **Specification / Property / Version** | **Description of the tool application** | **Availability** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Qualification Test Design Strategy

The purpose of the software qualification test is to ensure that the software is tested to provide evidence for compliance with the software requirements including functional safety requirements.

|  |  |
| --- | --- |
| **Test level** | Software Qualification test |
| **Test object** | Software function |
| **Test basis** | Software Requirements Specification including functional safety requirements |
| **Test Objective** | Carry out a series of tests on the whole software to assure the coverage of all software requirements and functionality for the given release |
| **Test method context** | Black box testing |

### Qualification test case design method activities

**For non-functional safety related test** at the software qualification test level, test cases can be derived using an appropriate combination of methods, as listed in the table below :

|  |  |
| --- | --- |
| **Test Design Method activities** | **Description** |
| Equivalence class generation and analysis | Divide a large dataset into different ranges where the testing effects are similar, making each range a representative group. Then, pick specific data points from each range to reflect the test results within that range. |
| Analysis of boundary values | The boundary value analysis method is a black box test method for testing the input or output boundary value. Check whether the upper, lower, and off points of the data can be processed correctly. |
| Causality graph method | Use a graphical method to illustrate different input combinations and then create a decision table to design the corresponding test cases. |
| Error guessing based on knowledge or experience | During program testing, individuals can use their experience and knowledge to anticipate potential errors in the program and develop specific test methods to check for these errors. |

**For functional safety testing** at the software qualification test level, test cases can be derived using an appropriate combination of methods, which listed in the table15 Methods for deriving test cases for integration testing of ISO26262-6:2018(E) should be considered.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Design Method activities** | **Description** | **ASIL** | | | |
| **A** | **B** | **C** | **D** |
| Analysis of requirements | Analyze the software requirements, functional safety to develop relevant test case. | x | x | x | x |
| Generation and analysis of equivalence class | Divide a large dataset into different ranges where the testing effects are similar, making each range a representative group. Then, pick specific data points from each range to reflect the test results within that range. |  | x | x | x |
| Analysis of boundary values | The boundary value analysis method is a black box test method for testing the input or output boundary value. Check whether the upper, lower and off points of the data can be processed correctly |  |  | x | x |
| Error guessing based on knowledge or experience | During program testing, individuals can use their experience and knowledge to anticipate potential errors in the program and develop specific test methods to check for these errors. |  |  | x | x |
| Analysis of functional dependencies | Whether the software functions meet customer requirements, new and changed functions should be covered |  |  | x | x |
| Analysis of operational use cases | Analyze the operation process and complete the use case design |  | x | x | x |

### Qualification test case execution method categories

|  |  |  |
| --- | --- | --- |
| **Execution Method** | **Description** | **Relevant for the project** |
| Requirement-based test | Software requirements and specific test requirements are used for testing. This includes testing related to cybersecurity and functional safety requirements. When it comes to cybersecurity requirements, test cases are designed based on attack patterns or STRIDE threat modeling. Additionally, cybersecurity test cases should also be created to verify the effectiveness of cybersecurity controls or measures | <Yes/no + justification if no> |
| Fuzz Test | Fuzz testing, or fuzzing, is a method in software testing where you intentionally feed incorrect or random data (called "fuzz") into a software system. The goal is to **find coding errors and security vulnerabilities**. In fuzz testing, the emphasis is primarily on the data that comes from external sources or interfaces of the software. You provide a wide range of valid and invalid input data, and you also randomize the timing of signals. Then, you check how the software behaves under these different conditions to identify potential issues. | <Yes/no + justification if no> |
| Penetration Test | Penetration testing assesses the security of an application to uncover vulnerabilities. It's crucial to confirm the **software's resistance to cyberattacks** before system-level testing. This involves verifying essential cybersecurity functions like key management, secure configuration, secure boot, software updates, secure communication channels, memory access rights, diagnostic services, and protection against brute force attacks. | <Yes/no + justification if no> |
| Fault injection test | Tests related to software injection into hardware or other interface faults (including the functional safety fault） | <Yes/no + justification if no> |

### Qualification test cases content

The test cases specification must contain the following information:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **ID** | Unique test case ID |
| **Title** | Title of the test case |
| **Status** | Indicate if the test case is Active / Not Active |
| **Level** | **Qualification test level** |
| **Purpose** | Describe the purpose of the test case |
| **Priority** | Indicate if the test case if of High, Medium, or Low priority for it’s execution |
| **Environment** | Indicate the test environment for the test case |
| **Method** | Describe the method that will be used for conducting the test case (see section 6.5.2) |
| **Pre-condition** | Define the condition(s) that have to be met to start the test |
| **Functional module** | Indicate the functional module to which the test case belongs |
| **FuSa** | Indicate if the test case is Functional Safety relevant |
| **Cybersecurity** | Indicate if the test case is Cybersecurity relevant |
| **Steps** | Create as many steps as required |
| **Steps titles** | Title of each step of the test case |
| **Steps descriptions** | Describe each step of the test case |
| **Steps expected results** | Describe the expected result of each step of the test case |

### Qualification test case priority

|  |  |
| --- | --- |
| **Test priority** | **Description** |
| **High** | Basic functions of the software, that affects the entire software function or cybersecurity and functional safety function |
| **Medium** | Basic functions of the module, tests that affect the module function. |
| **Low** | Small area of influence, could continue to perform tests on other functional modules |

Note: Basic functionality is the functionality that affects the overall software. if there is a problem with this functionality, the testing of the entire software functionalities will not be possible.

### Qualification test criteria

|  |  |
| --- | --- |
| **Test completion criteria** | There are a set of releases in the project. For each release a specific set of software requirements are delivered. The test completion applies the test objectives in section 2.7 only for the elements that need to be covered for this specific release. |
| **Test start criteria** | Complete module integrated software shall be available without any error and warnings, and source code should compile with test environment |
| **Test end criteria** | All test objectives for software qualification testing are achieved according to the section 2.7 test objectives. In addition, all bugs should be closed and all expected and actual results are captured and documented with the test script, 100% of software requirements shall link to test cases |
| **Test abort criteria** | if any stopper bug/blocker arises, testing shall be aborted |
| **Test restart criteria** | if any stopper bug/blocker arises, testing shall be aborted and after fixing bug, testing shall be restarted |

## Qualification test case/step result defect severity

For detail about software defect tracking, please refer to the [SUP.9 – Problem Resolution Management Process](https://confluence.engine.capgemini.com/x/vKtVAQ).

|  |  |
| --- | --- |
| **Defect severity** | **Description** |
| **Blocker** | Critical defects causing system or application crashes, freezes, system hangs, or loss of data, complete loss of major functions, cybersecurity function or functional safety abnormal |
| **High** | A major defect refers to the failure to implement a function or feature, the loss of a major function, causing a serious problem. |
| **Normal** | Less serious errors. Although such defects do not affect the basic use of the system, but if not better. |
| **Minor** | Some minor problems have little effect on the function, and the product and attributes can still be used, such as individual typos and irregularly arranged text. |

## Qualification testing Regression Test Strategy

Regression test shall verify that implemented SW changes have no unintentional "side effects" on software function that have not been changed.

A regression test shall be performed on basic functionality and safety relevant implementations.

During regression testing with unchanged test cases, test results should be as described below:

1) Unchanged test cases running on unchanged code shall "PASS"

2) Unchanged test cases running on changed code shall "FAIL"

3) Changed test cases on changed code shall "PASS"

The Regression testing shall include re-running of previously completed tests and to verify fixture of current reported defects and re-emergence of previously fixed defects. Regression test shall always be performed when any change is done in Software requirement for respective modules in every milestone of the project.

# Process tailoring

< Update If applicable >