**Table of contents**

**1 Objectives / Purpose 4**

**1.1 Deliverable Audience 4**

**2 Scope 5**

**3 Test Planning 6**

**3.1 Test Items 6**

**3.2 Test Scope 8**

**3.3 Risks, Assumptions and Constraints 9**

**3.4 Stakeholders and Communication 10**

**3.5 Resources and Responsibility 10**

**3.6 Testing Activities 11**

**3.6.1 Test Execution Completion Criteria 12**

**3.6.2 Test Execution Suspension Criteria and Resumption Requirements 13**

**3.7 Schedule 13**

**4 Testing Approach and Methodology 14**

**4.1 Testing Types and Levels 14**

**4.1.1 Unit Testing 15**

**4.1.1.1 Testing Types 16**

**4.1.1.1.1 Static code analysis 16**

**4.1.1.1.2 Configuration Tests 16**

**4.1.1.1.3 Structural Tests 17**

**4.1.1.1.4 Resource Usage Analysis 17**

**4.1.1.1.5 Unit Functional Tests 17**

**4.1.1.1.6 Unit Regression tests 18**

**4.1.2 Integration Testing 19**

**4.1.2.1 Testing Types 20**

**4.1.2.1.1 Fault Injection Tests 20**

**4.1.2.1.2 Interface Tests 21**

**4.1.2.1.3 Profiling tests 21**

**4.1.2.1.4 Compile only Tests 22**

**4.1.2.1.5 Stress Tests 22**

**4.1.2.1.6 State Transition Tests 23**

**4.1.2.1.7 Integration Regression tests 23**

**4.1.3 Qualification Testing 24**

**4.1.3.1 Testing Types 25**

**4.1.3.1.1 Functional Tests 25**

**4.1.3.1.2 Qualification Regression tests 26**

**4.1.4 Package / Release Testing 27**

**4.1.4.1 Package Verification Tests 27**

**5 Test Case Management 27**

**5.1 Test Design Techniques 27**

**5.2 Test Suites Composition 28**

**5.3 Grouping Test Cases to Test Suites 28**

**5.4 Test Case Traceability 28**

**6 Test Environment Requirements 29**

**7 Testing Deliverables 29**

**8 Acceptance Reviews and Approvals 31**

**9 Annexes 31**

**9.1 List of Tables 31**

**9.2 Test Type description 32**

**9.2.1 Requirements Based Testing 32**

**9.2.2 Equivalence Classes 32**

**9.2.3 Boundary Testing 32**

**9.2.4 Static Analysis 32**

**9.2.5 Fault Injection 32**

**9.2.6 Error Guessing 32**

**9.2.7 Performance Test 33**

**9.3 Test Specifications and Traceability 33**

**10 Document Information 33**

**10.1 References 33**

**10.2 Terms/Acronyms and Definitions 35**

**10.2.1 Standard Terminology 35**

**10.2.2 Terms/Acronyms and Definitions 37**

**10.3 Revision History 38**

# Objectives / Purpose

This document describes the overall test planning, design and execution of the SmartTRX family.

* Defines how softwaretesting of SmartTRX family is performed (testing methodology in terms of test levels, test types and test case derivation methods)
* Defines the test case development, test case execution and testing deliverable
* Defines the test scope for BL-RFP across various software subsystems derived from SmartTRX family software architecture specification refer section 10.1. Table-1 refers to the number variants derived out of SmartTRX family and identified software subsystems applicable across the derived variants

|  |  |  |  |
| --- | --- | --- | --- |
| **STRX Variant**  ***SW Subsystem*** | STRX One-Chip | STRX Remote | STRX SHARK |
| *Application Bootloader* |  |  | *Not Applicable* |
| *Platform SDK* |  | customized | *Not Applicable* |
| *Safety SDK* |  |  |  |
| *Radar SDK* |  |  | *Not Applicable* |
| *RFE software* |  |  |  |

Table 1 SmartTRX software subsystem

Details about communication mechanism between the controllers and host system in case of STRX Remote and STRX SHARK are yet to be defined, similarly the details about the customized platform SDK in STRX Remote are yet to defined. Based on the definition of these aspects the test objective for the STRX Remote and STRX SHARK variants will be detailed during PPA

### **Deliverable Audience**

The SmartTRX Test plan is intended for the following audience:

BL-RFP/BL-AP Integration and Validation teams, BL-RFP/BL-AP software development teams, Integration and Configuration management teams, System Integration Team, Project managers, Safety managers, Quality Assurance team members

# Scope

The scope of the software test plan includes all incremental testing refer section 3.6 to be conducted at various milestones which are defined in software project management plan [040.20\_SmartTRX\_SW\_PMP] refer section 10.1. Below listed are top level software testing scope identified for SmartTRX project,

In scope:

* Test plan covers software test activities to be conducted for STRX One-chip variant as of now. The details for the STRX Remote and STRX SHARK variants will be added to this document or captured separately during PPA.
* Covers usage of Chirp design tool for software validation of RFE APIs.
* Covers enablement software which includes mandatory requirements listed for application bootloader, platform SDK, safety SDK & RFE software in STRX One-chip variant to be developed and validated by Early Access Release. Refer to software project management plan [040.20\_SmartTRX\_SW\_PMP] from section 10.1
* Covers limited testing needs of open system architecture by means of verifying low level hardware register access interfaces which are implicitly covered as part of platform, radar, safety SDK & RFE software verification exercise. Refer constraint section 3.3.3 for open system architecture testing limitation
* Covers testing of radar reference application software part of platform integration which is provided for demonstration to customer and needed by “Release for production” (RFP) milestone
* Covers all 3rd party software deliveries (Software deliverable from BL-AP, FreeRTOS) listed under section 3.5 which shall be validated and verified by supplier and this document will define the acceptance criteria for 3rd party software deliverable
* Covers Qualification testing of primary bootloader [ROM code] on simulation environment (VDK or ZeBu) and FPGA
* Covers reference application boot loader
* Covers testing of platform SDK which supports AUTOSAR and NON- AUTOSAR software configurations
* Covers testing of all SDKs which are defined in SmartTRX software architecture specification

Out of scope:

* Does not cover validation of test tools. This is in scope of functional safety team which assesses test tool and their qualification reports[1]
* Does not cover test cases covering detections in hardware. This is in scope of global validation team
* Does not cover validation of security software that runs on HSE subsystem, because it requires trust centre tools. This is in scope of BL-RFP Security team [3]
* Does not cover validation of crypto libraries which are placed in the boot ROM. This is in scope of BL-RFP Security team [3]
* Does not cover functional characterization using RFE calibration interfaces including the chirp design tool. This is in scope of global validation team and software verification shall check outcome till register [2]
* Does not cover validation of boot ROM, OTP content and product life cycle test. This is in scope of global validation team [2]
* Does not cover validation of advanced radar algorithms. This is in scope of global validation team [2]
* Does not cover validation of safety manager configurations. This is in scope of global validation team [2]

[1] - *Aligned with BL-RFP SW architects*

[2] - *Aligned with global validation team*

[3] - *Aligned with BL-RFP SECURITY architects*

# Test Planning

### **Test Items**

- Software deployment across various cores APP-M7, RFE-M7, SPT-BBE & APP-BBE are detailed in SmartTRX architecture specification refer section 10.1

- SmartTRX test item constitutes the software drivers, algorithms, operating system and integration needs across software subsystem are listed below,

1. Application Bootloader: This constitutes boot loader which runs on APP-M7 core

* CAN Driver, QUAD-SPI Driver, MCAL libraries, self-test

1. Platform SDK: This constitutes MCAL, non MCAL, AUTOSAR OS and connectivity drivers which runs on APP-M7 core

* High Speed IF Cluster: Ethernet/SGMII Driver, AURORA driver/Tooling
* AUTOSAR Stack:

1. Complex Device Drivers : I2C, MCL, DAC, TSENSE, XRDC, CORE
2. AUTOSAR compliant drivers:

* Communication drivers: CAN, SPI, Ethernet
* Memory Drivers: FLS, FEE
* Microcontroller drivers: GPT, MCU, WDG
* Security drivers: Crypto Driver (only compilation and no functional test)
* I/O Drivers: DIO, PWM, ICU, OCU, ADC, PORT

1. Operating System: AUTOSAR OS

* NON AUTOSAR Stack:

1. Platform Non-MCAL Drivers:

* Peripheral drivers: Timer, WDG, Pin-Pad control, GPIO, DMA, UART, Interrupt Handling, System/Soc Init, Debug
* Peripheral Abstraction Layer

1. Operating System: FreeRTOS
2. Safety SDK: This constitutes of software that enables to achieve safety integrity level ASIL-B which runs on APP-M7 core

* Primary Bootloader safety tests LBIST and MBIST.
* Safety manager on APP-M7.
* Embedded functional safety of RFE-M7 and APP-BBE also considered
* Square check results from different cores: SPT-BBE, APP-BBE and RFE-M7
* RFE ISM, functional safety tests.

1. Radar SDK: This constitutes of the radar data processing software that runs on the SPT, SPT-BBE, APP-BBE and the application software runs on APP-M7 core

* SPT Driver operation such as init/run/cmd/stop which shall be executed in application core
* SPT kernels which are available as assembly code and invoked by SPT driver through relevant function prototypes. SPT kernel refers to a series of instructions to perform a radar processing function
* SPT dispatcher software handler behaviour on reception of radar frame signal from RFE subsystem
* BBE DSP kernels/functions invoked by the STP IP (or application directly)

1. RFE Software: This constitutes of the software which runs on RFE-M7 and any client software runs on APP-M7 core

* RFE driver that gets executed on APP-M7: RFE client APIs, RPC client
* RFE software that gets executed on RFE-M7: RFE server APIs, Communication drivers

1. Platform Integration: This is not part of software subsystem but enables the integration of other subsystem and demonstrates smartTRX reference application as specified in software architecture specification refer section 10.1

* Any software related to communication interfaces such as ethernet stack, LVDS TX Driver, CSI-2 which are not covered under PLATFORM SDK
* IPC related software stacks which are not covered under PLATFORM SDK
* Infra drivers for APP-BBE/RFE-M7(if applicable) core
* Use case level testing for advanced radar algorithms that runs on APP-BBE
* Algorithms implemented on SPT-BBE for SmartTRX
* Kernels implemented on SPT

### **Test Scope**

* This section specifies the features of test item what is in-testing scope and what is out-of-testing scope for STRX One-chip variant.
* Refer section 3.6 for the detailed testing activities to be conducted on the features of test item being identified.
* Features listed below are derived from software requirement specification refer section 10.1. The feature list has the must have requirement of smartTRX project and this section will be enhanced during PPA
  + 1. **In Scope:**

1. Application Bootloader:

* Testing of application boot loader sequence on all the cores

1. Platform SDK:

* Testing of AUTOSAR configuration including OSEK
* Testing of NON-AUTOSAR configuration including FreeRTOS
* Testing of Software common chassis – drivers in FreeRTOS context
* Testing of infra drivers on APP-BBE

1. Safety SDK:

* Testing of Functional Safety manager
* Testing of functional safety mechanism
* Testing of operation modes during and after booting in secure and non-secure modes
* Testing of data flow and control flow tests on APP-BBE and SPT-BBE
* Testing of Functional Safety at runtime, from normal to degraded mode

1. Radar SDK:

* Testing of SPT kernel configuration by SPT driver
* Testing the Scheduling of SPT dispatcher in a system cycle
* Testing functional safety of SPT-BBE/SPT, APP-BBE
* Testing advanced radar algorithms (DoA, clustering, tracking)

1. RFE software:

* Testing of APIs Client and server side,

1. IP API
2. Stream API [DMA]
3. System API
4. Use case API

* Testing of Inner functional safety
* Testing of calibration
* Testing of Peripheral Test
* Testing of Use cases, this includes the main loop to process IPC commands
* Testing of system cycle loop (PLL and transmit /receive paths)
* Testing of Configuration of chirps (timing engine, phase rotator)
* Testing of RFE infra drivers

1. Platform Integration:

* Testing of IPCF
* Testing text book radar application

### **Risks, Assumptions and Constraints**

* + 1. **Assumptions:**

1. Platform SDK:
2. Existing test cases from BL-AP shall be reused and enhanced as needed by BL-RFP for reproducing customer reported issues [2]
3. Test framework support shall be provided by BL-AP and execution handled by BL-RFP [2]
4. During pre-silicon testing, BL-AP shall handle unit, integration testing of primary bootloader [2]
5. During pre-silicon and post-silicon testing, for technology specific HW IPs with BL-AP compatible APIs, unit and integration testing shall be performed by BL-AP and qualification testing with hardware shall be handled by BL-RFP [2]
6. During pre-silicon and post-silicon testing, for technology specific HW IPs with BL-AP incompatible APIs unit, integration and qualification testing with hardware shall be handled by BL-RFP [1]
7. During post-silicon testing, unit, integration and qualification testing with hardware shall be handled by BL-RFP [1]
8. Safety SDK:
9. Fault seeding/injection shall be performed in emulated environment or through code instrumentation for read hardware write register [3]
10. Coverage reports for safety routines are provided by Cadence and BL-AP [3]
11. Radar SDK:
12. Reference applications which covers use cases to be shared with Integration and Validation team [1]
13. RFE software:
14. Reference applications which covers use cases to be shared with Integration and Validation team [1]
15. RFE Analogue model developed in a simulated environment is provided [4]

[1] - *Aligned with BL-RFP SW architects*

[2] – *To be Aligned with BL-AP team*

[3] - *Aligned with BL-RFP Safety SW architects*

[4] – *Aligned with global validation team*

* + 1. **Risks:**

For detailed risk identified for the project are captured in software FMEA in the following path

[080.05\_SmartTRX\_FMEA\_SW](https://www.collabnet.nxp.com/sf/go/doc364212?nav=1&pagenum=1&pagesize=15)

1. Radar SDK:

* Limited to text book corner radar application software and to mitigate this performance tests will be defined and executed
  + 1. **Constraints:**

1. Only STRX one-chip hardware will be available for initial software development testing activity, other variants hardware and related software activity to be defined by PPA
2. Limited testing of open system architecture conducted since realization of customer actual field use case and system needs cannot be recreated in software testing.

### **Stakeholders and Communication**

* The Integration and Validation team communicates the activity status during project meetings defined in Software Project Management Plan [040.20\_SmartTRX\_SW\_PMP] refer section 10.1
* The Integration and Validation team will create the baseline of test deliverable refer section 7 and shares report to the entire project team (development team, project manager, quality assurance, safety manager)
* Stakeholder information is detailed in Software Project Management Plan [040.20\_SmartTRX\_SW\_PMP] refer section 10.1
* Release package creation for smartTRX variant shall be detailed in Software Configuration Management Plan [SCMP] refer section 10.1

### **Resources and Responsibility**

* Project organization information are captured in Software Project Management Plan [040.20\_SmartTRX\_SW\_PMP] refer section 10.1
* Roles and responsibility of various members involved in validation activity is detailed in BCAM AUTO Validation Procedure- BcaM 7.0 and can be referred from following link [NXPOMS-1719007347-3856](https://nxp1.sharepoint.com/:w:/r/sites/OMS/_layouts/15/WopiFrame.aspx?sourcedoc=%7b29780877-DBAC-4FF7-AA66-8415B7E2F064%7d&file=BU%20AUTO%20Validation%20Procedure%20-%20BCaM7.0.docx&action=default&DefaultItemOpen=1)
* Verification activity of Platform SDK, Radar SDK, Safety SDK are managed by BL-AP
* Verification activity of RFE software and application bootloader are managed by BL-RFP
* SmartTRX Platform Integration & overall verification activity are managed by BL-RFP

### **Testing Activities**

* This chapter is intended primarily for testers. It provides chronological listing of all test activities performed at different release milestones
* Detailed test schedule to be carried out for various release milestones with iterative releases identified for smartTRX project are detailed in the path below, <https://www.collabnet.nxp.com/sf/go/doc417653>
* Test specification shall include the detailed testing procedure to be conducted on the features of the test item identified
* Software testing process metrics to be achieved for various project milestones are detailed in SW quality assurance plan refer [070\_01\_SW Quality Assurance Plan](https://www.collabnet.nxp.com/sf/docman/do/downloadDocument/projects.smarttrx/docman.root.es0.070_project_quality_assurance/doc432165) under section Project Quality Goals
* Tester shall follow the guideline of sequential execution identified below for various milestones in smartTRX project. Refer section 4.1 for details regarding test types and test level

**Early Access Release (EAR):**

Software Validation Engineer shall primarily focus on application bootloader, platform SDK, safety SDK & RFE software in SmartTRX enablement software validation activities as following steps which shall be executed for the software test item identified for EAR milestone;

Step 1: Software Unit test specification which shall cover static code analysis, configuration tests, structural tests, resource usage analysis, unit functional tests and unit regression tests refer section 4.1.1

Step 2: Software integration test specification which shall constitute fault injection tests, interface tests, profiling tests, compile only tests, stress tests, state transition tests and integration regression test refer section 4.1.2

Step 3: Identified test case shall be executed and failure shall be reported through a ticket in the project JIRA database. Creation of JIRA ticket detailed in Software Configuration Management Plan [SCMP] refer section 10.1

Step 4: Enhancement or addition of missing test case after baseline shall be done through a ticket in the project JIRA database.

Step 5: Test report generated shall be checked in the repositories as defined in Software Configuration Management Plan [SCMP] refer section 10.1

**Production Release Candidate (PRC):**

Software Validation Engineer shall primarily focus on validation of SmartTRX enablement and Reference application software

Step 1: Qualification test specification which covers functional test and qualification regression test refer section 4.1.3

Step 2: Newly added software test items shall follow all the steps listed in EAR

Step 3: Unit and integration test for delta changes shall be executed along with regression testing

Step 4: Identified test case shall be executed and failure shall be reported through a ticket in the project JIRA database

Step 5: Showstopper or High priority failures shall be analyzed and fixed

Step 6: Enhancement or addition of missing test case after baseline shall be done through a ticket in the project JIRA database.

Step 7: Test report generated through automated test flow shall be checked in the repositories as defined in SCMP refer section 10.1

**Release For Production (RFP):**

Software Validation Engineer shall primarily focus on validation of SmartTRX product application software

Step 1: Newly added software test items shall follow all the steps listed in PRC

Step 2: During RFP complete automation framework for Unit, Integration and Qualification Regression test shall be deployed to enable previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made with limited manual efforts

Step 3: Reporting failed test through ticket in the project JIRA database

Step 4: Enhancement or addition of missing test case after baseline shall be done through a ticket in the project JIRA database.

Step 5: Test report generated through automated test flow shall be checked in the repositories as defined in SCMP refer section 10.1

#### Test Execution Completion Criteria

Specifies the criteria used to decide that test execution activities can be considered finished.

1. Analysis and correction of compile time errors and warnings in source code
2. Analysis and correction of MISRA violations
3. Analysis and documentation of cyclomatic complexity metrics
4. Analysis and documentation of the resources used per software driver (ROM for code and constant data, RAM for global static data)
5. Review of BL-AUTO Coding guidelines adherence, static code analysis report and Resource Usage Report
6. Verification of generated configuration files used in the project
7. Verification of basic functionality of each function according to the software design specification and Software Unit testing covers the complete source code of the software unit
8. Verification of correct implementation of functional requirements according to software architecture specification and verification of dependencies between APIs in software integration testing
9. Verification of correct implementation of functional safety software requirements
10. Verification of functional software requirements implemented in software during software qualification testing
11. Documentation of justification for all violations that cannot be fixed in the project
12. Validation of SmartTRX demo application
13. Review of traceability established across various artefacts in smartTRX project

#### Test Execution Suspension Criteria and Resumption Requirements

Specifies the criteria used to suspend all or a part of the testing execution activities.

1. Defect ticket shall be raised for any defects found during testing and appropriate priority level shall be assigned (High/Medium/low)
2. Defect ticket shall be raised for deviation in KPIs identified in quality assurance plan for SmartTRX project
3. Based on the impact change control board decides to suspend or continue to execute the planned test activities

### **Schedule**

1. **Test campaign** is the timeline within which detailed testing activities briefed in section 3.6 are performed
2. Typically, the test campaign duration is 2-6 weeks, depending on the maturity of the software development releases and test cases development
3. Test campaign can be repeated in several test cycles, typically during initial phase of development 3 complete test cycles to be planned and can be reduced to 1 test cycle after moving to complete automated test campaigns
4. STRX One-chip variant project schedule are captured in Software Project Management Plan [040.20\_SmartTRX\_SW\_PMP] refer section 10.1 to consider above inputs for planning of software releases

# Testing Approach and Methodology

The test levels mapping with the V-Model for STRX One-chip variant project is given in the following figure:

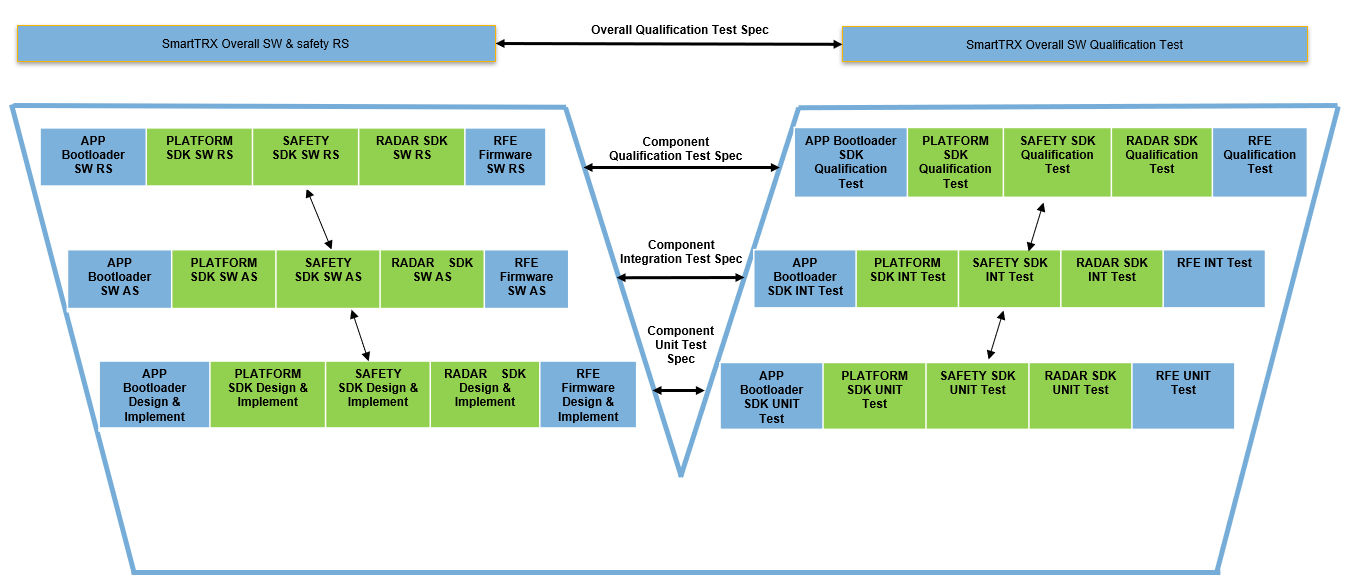


Figure 1**:** STRX One-chip variant V Model Graphical representation

*legend*: RFP Software Component Testing

AP Software Component Testing

### **Testing Types and Levels**

This Section defines test levels for STRX One-chip variant project,

Software testing levels are categorized further as Software Unit, Integration and Qualification [Validation] test as defined by IEEE Std. 610 1990

1. Unit Testing: The test of individual software driver to verify that the specified design has been correctly implemented
2. Integration Testing: The test performed to expose defects in the interfaces and interaction between integrated driver to verify that the specified architecture has been correctly implemented
3. Qualification/Validation Testing: The process of testing an integrated software product to validate that the specified requirements have been correctly implemented

The interactions between this testing level is captured in the Figure 1 and criteria for all these stages are detailed below

#### Unit Testing

* Software Unit Verification Process is to verify software units to provide evidence for compliance of the software units with the software detailed design and with the non-functional software requirements
* Unit test shall be performed on the individual test item captured for various software sub system in section 3.1.
* Software unit design requirements, interfaces, processing, data type definition and error handling detailed in the software detailed technical specification shall be verified during unit testing
* Unit testing activity shall be handled by the developers

**Process:**

1. software unit verification strategy including regression strategy is developed to verify the software units.

* Software unit test case created in accordance with unit test specification are deployed in LDRA tool and continuous integration tests are automated through BAMBOO. Build management defined in Software Configuration Management Plan [SCMP] shall be followed for continuous integration

1. criteria for software unit verification are developed according to the software unit verification strategy that are suitable to provide evidence for compliance of the software units with the software detailed design and with the non-functional software requirements

* Traceability established in REQTRACER across software design document to that of UNIT test specification
* Boundary value Equivalence class derived for software units

1. software units are verified according to the software unit verification strategy and the defined criteria for software unit verification and the results are recorded

* On completion of test execution, the test reports are stored in predefined path specified in configuration management for various releases

1. Consistency and bidirectional traceability are established between software units, criteria for verification and verification results

* Traceability established across Software Design Document, Test specification and Test report. Violation agreed for specific release shall be documented and be part of traceability

1. results of the unit verification are summarized and communicated to all affected parties

* Issues are logged in JIRA and assigned to developer for analysis
* Test reports and progress are shared to relevant stakeholder
* Review conducted for test artefacts shall be stored in collabnet and bitbucket

##### **Testing** **Types**

* + - * 1. **Static code analysis**

The source code is statically verified by running static code analysis tool to identify violations of the MISRA rules and by performing peer reviews of the code.

Objective: identify violations of the MISRA rules; verify if the implementation is according to the detailed design and to the Coding Style.

Input:

1. Coding guidelines, including applicable MISRA rules
2. Driver source code [SW unit]

* Source code (.c and .h)

Test Methods:

* Run static analysis tool/ common weakness enumeration (CWE)
* Code review

Deliverables:

* MISRA status report [Refer section **Testing Deliverables**]
* Cyclomatic Complexity reports [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]
  + - * 1. **Configuration Tests**

The scope of this code generation test at Unit Test level is verification of generated configuration files as defined in Software design specification.

Input:

* 1. software design document
  2. Configuration Files Suite
* Configuration files (e.g.,xdm and/or .bmd files, macro files, property files, plugin.xml files)
* Code generator template files (e.g. \*cfg.c and \*cfg.h)
* Configuration tool – EB Tresos

Test Methods:

* Analysis of requirements
* Analysis of structure (statement, decision, multiple condition coverage)
* Generation and analysis of equivalence classes
* Analysis of boundary values
* Error guessing

Deliverables:

* Test Report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]
  + - * 1. **Structural Tests**

Structural testing is based upon the structure of code or data. The method used is white box testing.

Input:

* + - * 1. software design document
        2. Driver source code [SW unit]
* Source code (.c and .h)

Test Methods:

* Analysis of requirements
* Generation and analysis of equivalence classes
* Analysis of boundary values
* MC/DC Coverage
* Error guessing

Deliverables:

* Code coverage reports [Refer section **Testing Deliverables**]
* Test Report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]
  + - * 1. **Resource Usage Analysis**

Analysis and documentation of the resources used per software Driver (ROM for code and constant data, RAM for global static data)

Input :

software design document

Driver source code [SW unit]

* Driver source code (.c and .h)

Test Methods:

* Performance test

Deliverables:

* Code Size report [Refer section **Testing Deliverables**]
  + - * 1. **Unit Functional Tests**

Functional testing are tests, which are executed on the target. White Box tests serves for verification of unit test design. White box tests are created with the knowledge of component design and its implementation. White box tests can use also underlying layers and they can directly access processor registers or memory.

Input:

1. software design document
2. Driver source code [SW unit]

* Source code (.c and .h)

Test Methods:

* Analysis of requirements
* Test suite mapped at software unit

Deliverables:

* Test Specification [Refer section **Testing Deliverables**]
* Test source files: test cases, test suites, test vectors [Refer section **Testing Deliverables**]
* Traceability matrix [Refer section **Testing Deliverables**]
* Test Reports [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]
  + - * 1. **Unit Regression tests**

Unit Regression testing is retesting a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made in software units. All existing and the newly added that are executed on the target HW and that can be automated in the automated environment become part of the regression test suite.

Input:

* Release test plan [Refer section **Testing Deliverables**]

Test Methods:

* Test suite mapped at software unit

Deliverables:

* Test Specification [Refer section **Testing Deliverables**]
* Test source files: test cases, test suites, test vectors [Refer section **Testing Deliverables**]
* Traceability matrix [Refer section **Testing Deliverables**]
* Test Reports [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test types**  ***SW Subsystem*** | Static code analysis | Configuration Test | Structural Testing | Resource Usage Analysis | Unit Functional Tests | Unit Regression Test |
| *Application Bootloader* |  |  |  |  |  |  |
| *Platform SDK* |  |  |  |  |  |  |
| *Safety SDK* |  |  |  |  |  |  |
| *Radar SDK* |  |  |  |  |  |  |
| *RFE software* |  | *Not Applicable* |  |  |  |  |

Table 2 Unit level Test type mapping to SW subsystem

*legend*: EAR milestone

PRC milestone

RFP milestone

#### Integration Testing

* The purpose of the Software Integration and Integration Test Process is to integrate the software units into larger software items up to a complete integrated software consistent with the software architectural design
* To ensure that the software items are tested to provide evidence for compliance of the integrated software items with the software architectural design, including the interfaces between the software units and between the software items
* External interfaces, Typical usage, performance requirements, major internal interfaces, internal software states, performance issues, Dependencies detailed in the software architecture specification shall be tested in integration testing
* Integration testing will be handled by Integration and Validation team

**Process:**

1. Software integration test strategy including the regression test strategy is developed to test the software unit and software item interactions

* Software integration test case created in accordance with integration test specification are deployed in automation test framework and continuous integration test are automated through BAMBOO. Build management defined in Software Configuration Management Plan [SCMP] shall be followed for continuous integration

1. Specification for software integration test according to the software integration test strategy is developed that is suitable to provide evidence for compliance of the integrated software items with the software architectural design, including the interfaces between the software units and between the software items

* Traceability established in REQTRACER across software architecture document to that of Integration test specification

1. Software units and software items are integrated up to a complete integrated software according to the integration strategy

* Integration test specification to define the dependency driver information, boundary value and equivalence class ranges for software unit integration

1. Test cases included in the software integration test specification are selected according to the software integration test strategy, and the release plan

* Regression Test suite and case shall be classified as functional, nonfunctional and only relevant test suite shall be triggered, and scope shall be specified in the standard test release plan

1. Integrated software items are tested using the selected test cases and the results of software integration test are recorded

* On completion of test execution, the test reports are stored in predefined path specified in configuration management for various releases

1. Consistency and bidirectional traceability are established between the elements of the software architectural design and the test cases included in the software integration test specification and between test cases and test results
2. Results of the software integration test are summarized and communicated to all affected parties

* Test progress shall be broadcasted to change control board and relevant stake holders
* Review conducted for test artefacts shall be stored in collabnet and bitbucket

##### **Testing Types**

###### **Fault Injection Tests**

In accordance to ISO 26262-6 for safety related software, fault injection tests during integration testing are used to verify all safety and error handling mechanisms on the target HW, and to demonstrate their effectiveness to detect and handle systematic SW failures or random HW and system failures. Fault injection tests include:

* directed fault injection to trigger all safety mechanisms,
* directed fault injection to trigger all standard error handling mechanisms (e.g. DEM/DET)

Fault injection testing may require interface stubbing, or even instrumentation of the code under test.

Input:

* 1. software subsystem Architectural Specification
  2. source code of all SW units or the integrated software driver
* Source code (.c and .h)
* Configuration files (.xdm and .bmd files, macro files, property files, plugin.xml files) and configuration templates (\*cfg.c and \*cfg.h), if applicable

Test Methods:

Verification of complete source code of all functions (APIs):

* Analysis of safety requirements
* Verification of the specified / intended functionality
* Verification of effectiveness of safety and error handling mechanism
* Error guessing

Deliverables:

* Test report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

###### **Interface Tests**

Testing of a software with the intent to determine how well the software unit interacts with one or more specified software unit it should interact with.

Input:

* 1. software subsystem Architectural Specification which indicates the dependant information
  2. Source code of all SW units or the integrated software driver
* Source code (.c and .h)
* Configuration files (.xdm and .bmd files, macro files, property files, plugin.xml files) and configuration templates (\*cfg.c and \*cfg.h), if applicable

Test Methods:

* Verification of the specified / intended functionality
* Verification of different configurations in configuration tool
* Verification of interoperability of the implementation
* Generation and analysis of equivalence classes
* Analysis of boundary values

Deliverables:

* Test report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

###### **Profiling tests**

The purpose of profiling test is to measure duration of execution of individual APIs.

Input :

* 1. software subsystem Architectural Specification
  2. Driver source code [SW unit]

- Source code (.c and .h)

Test Methods:

* Performance test

Deliverables:

* Profile report [Refer section **Testing Deliverables**]

###### **Compile only Tests**

Compile only testing is intended to check correctness of statically configured code, i.e. to check different combination of parameter values and to expose potential problems with generation of the driver configuration source code and compilation of generated source code.

Input:

* 1. software subsystem Architectural Specification
  2. Driver source code [SW unit]
* Source code (.c and .h)
* Configuration files

Test Methods:

* Analysis of requirements
* Generation and analysis of equivalence classes
* Analysis of boundary values
* Error guessing

Deliverables:

* Test specification [Refer section **Testing Deliverables**]
* Test report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

###### **Stress Tests**

A test type concerned with measuring the behaviour of a component or system with increasing load, e.g. number of parallel users and/or numbers of transactions to determine what load can be handled by the component or system. stress tests need to exercise all resources, but does not need to check complete functionality available on all resources.

Input :

* 1. software subsystem Architectural Specification
  2. Driver source code [SW unit]

- Source code (.c and .h)

Test Methods:

* Performance test

Deliverables:

* Test specification [Refer section **Testing Deliverables**]
* Test report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

###### **State Transition Test****s**

State Transition is to be used in any of the following cases:

1. Software architecture specification of the system identifies behavioral states in which it resides
2. Software architecture specification contains state tables, state transition model or state transition diagrams

is applied to exercise tests that verify that invalid transitions cannot occur, and that pay attention to state recovery after unpredictable events (e.g. power-fail/power-up recoveries)

Input:

* 1. software subsystem architectural specification
  2. Driver source code [SW unit]

Test Methods:

* Analysis of requirements
* Generation and analysis of equivalence classes
* Analysis of boundary values
* Error guessing

Deliverables:

* Test specification [Refer section **Testing Deliverables**]
* Test report [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

###### **Integration Regression tests**

Integrated Regression testing is retesting a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made in integrated software units. All existing and the newly added that are executed on the target HW and that can be automated in the automated environment become part of the regression test suite.

Input:

* Release test plan[Refer section **Testing Deliverables**]

Test Methods:

* Test suite mapped for integrated software units

Deliverables:

* Test specification [Refer section **Testing Deliverables**]
* Test report [Refer section **Testing Deliverables**]
* Test source files: test cases, test suites, test vectors [Refer section **Testing Deliverables**]
* Traceability matrix [Refer section **Testing Deliverables**]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test types**  ***SW***  ***Subsystem*** | Fault Injection Tests | Interface Tests | Profiling Tests | Compile only tests | Stress tests | State Transition tests | Integration regression tests |
| *Application Bootloader* | Not applicable |  |  |  |  |  |  |
| *Platform SDK* |  |  |  |  |  |  |  |
| *Safety SDK* |  |  |  |  |  |  |  |
| *Radar SDK* |  |  |  |  |  |  |  |
| *RFE software* |  |  |  |  |  |  |  |

Table 3 Integration level Test type mapping to SW subsystem

*legend*: EAR milestone

PRC milestone

RFP milestone

#### Qualification Testing

* The purpose of the Software Qualification Test Process is to ensure that the integrated software is tested to provide evidence for compliance with the software requirements
* Software functional requirements detailed in the software requirement specification and which are not of review type shall be tested in qualification testing
* Qualification testing will be handled by Integration and Validation team

**Process:**

1. Software qualification test strategy including regression test strategy consistent with the project plan and release plan is developed to test the integrated software

* Build management defined in Software Configuration Management Plan [SCMP] shall be followed for continuous integration

1. Specification for software qualification test of the integrated software according to the software qualification test strategy is developed that is suitable to provide evidence for compliance with the software requirements
2. Consistency and bidirectional traceability are established between software requirements and software qualification test specification including test cases and between test cases and test results

* Traceability established in REQTRACER across software architecture document to that of qualification test specification

1. Test cases included in the software qualification test specification are selected according to the software qualification test strategy and the release plan

* Regression Test suite and case shall be classified as functional, nonfunctional and only relevant test suite shall be triggered, and scope shall be specified in the test release plan

1. Integrated software is tested using the selected test cases and the results of software qualification test are recorded

* On completion of test execution, the test reports are stored in predefined path specified in configuration management for various releases

1. Results of the software qualification test are summarized and communicated to all affected parties

* Test progress shall be broadcasted to change control board and relevant stake holders
* Review conducted for test artefacts shall be stored in collabnet and bitbucket

##### **Testing Types**

###### **Functional Test****s**

Functional testing are tests, which are executed on the target. Black Box tests validate component functionality as specified in software requirements. Black Box tests are created only with API interfaces, which are used to interact with the component. Black Box tests are created without the knowledge of component implementation and without use of processor register or memory.

Input:

* 1. software subsystem requirement specification
  2. Driver source code [SW unit]
* Source code (.c and .h)

Test Methods:

* Analysis of requirements
* Test suite mapped at integrated software product

Deliverables:

* Test Specification [Refer section **Testing Deliverables**]
* Test source files: test cases, test suites, test vectors [Refer section **Testing Deliverables**]
* Traceability matrix [Refer section **Testing Deliverables**]
* Test Reports [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

###### **Qualification Regression test****s**

Qualification Regression test is retesting a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made in release candidate module. All existing and the newly added that are executed on the target HW and that can be automated in the automated environment become part of the regression test suite.

Input:

* Release test plan[Refer section **Testing Deliverables**]

Test Methods:

* Test suite mapped for the integrated software product

Deliverables:

* Test Specification [Refer section **Testing Deliverables**]
* Test source files: test cases, test suites, test vectors [Refer section **Testing Deliverables**]
* Traceability matrix [Refer section **Testing Deliverables**]
* Test Reports [Refer section **Testing Deliverables**]
* Inspection review record [Refer section **Acceptance Reviews and Approvals**]

|  |  |  |
| --- | --- | --- |
| **Test types**  ***SW***  ***Subsystem*** | Functional Tests | Qualification Regression tests |
| *Application Bootloader* |  |  |
| *Platform SDK* |  |  |
| *Safety SDK* |  |  |
| *Radar SDK* |  |  |
| *RFE software* |  |  |

Table 4 Qualification level Test type mapping to SW subsystem

*legend*: EAR milestone

PRC milestone

RFP milestone

#### Package / Release Testing

##### **Package Verification Tests**

Integrated software products are released together as a package. The package is released with a demo application which is meant to provide an example of how to use the integrated software product.

# Test Case Management

This section covers the test suites and test cases management for smartTRX project

### **Test Design Techniques**

This chapter describes test techniques tailored for SmartTRX testing, used to develop new test scenarios, test vectors and test configurations

**Test scenario** is step by step procedure describing actual implementation of the test case, i.e. primarily sequence of API calls and assertions describing pass/fail conditions. Test scenario includes also preconditions for HW setup. Test scenarios are created primarily using **Requirements based testing, and error guessing.**

**Test specification** contains **test scenarios** and **test classification**. Test specification does not describe **test vectors** – see definition below. Test specification source is stored in the test case implementation in form of comments with dedicated Doxygen tag. Test specification document is generated from source codes of test cases and test suites for each release in the end of each test activity completion for the release.

**Test vector** is a set of values used as input for the **test scenarios** and it includes also expected values which are compared in the test with output values of tested entity or with values of processor registers or driver internal variables. Test vectors could be contained directly in the test case file or they can be included from external files, to facilitate automated test vector generation.

**Test case** is a pair of a test scenario and a test vector. Test cases are implemented in C language, by default one file per test case. Each test case includes its test specification. Test cases shall be written in way that they can be executed in any order. All exception from this rule needs to be documented in the test specification.

**Test case set** is test scenario, which exercises multiple test vectors. In the test report the test case set appear as the same as test case – only 1 test result exists for all test vectors (single failure means failure of complete test case set).

**Test suite** aggregates individual test cases and test case sets. It is created by common test suite source code. Each test suite has its make file describing all dependencies. **Only test suite is executable**, no test case can be executed independently.

A test suite shall contain detailed instructions or goals for each collection of test cases and information on the system configuration to be used during testing. Test suite is always treated as specific to certain release, i.e. since it defines applicable subset of generic or specific test cases.

### **Test Suites Composition**

Test suites and test cases creation are described in Figure 2

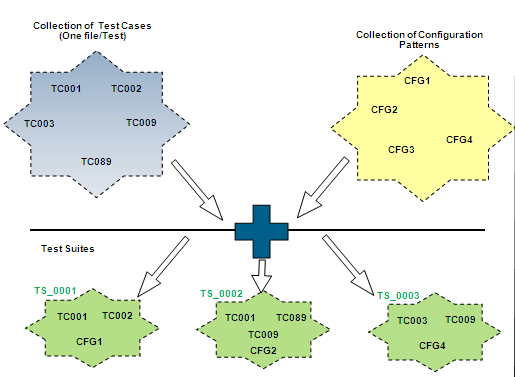


Figure 2: Test Suites Creation

### **Grouping Test Cases to Test Suites**

* Test cases are grouped into test suites, when they can share certain configuration or set of configurations.
* If precompiled configuration exists, that suits the prerequisites of the test case, then it shall be used mandatorily in order to **keep minimum number of test suites** (test application flashing brings significant overhead during automated test execution).

### **Test Case Traceability**

Reqtracer traceability project establishes the vertical and horizontal traceability across various artefacts by establishing mapping of unique IDs used in the project.

* Test coverage established across requirement, architectures and design
  + Traceability Analysis report
* Presence of anomalies such as:
  + References to non-existing requirements or Design IDs.
  + Duplicated requirements
  + Duplicated Design IDs

# Test Environment Requirements

Test environment comprises of software tools, hardware items and automation framework which are listed in SmartTRX overall software architecture under Tools section (2.1.7)

<https://www.collabnet.nxp.com/sf/go/doc347357?nav=1&pagenum=1&pagesize=15>

# Testing Deliverables

Integration and validation team deliverables identified for the STRX One-chip variant are listed below,

| **Test Deliverable** | **How to create it** |
| --- | --- |
| Release Test plan | **Release test plan** is with all test cases, generated by script, and manually filled in applicability of each test to test platform/release.  release Test plan template constitutes,  a. Mapping between test suite and test case  b. Mapping to appropriate Test type, test level, test technique  c. Mapping to release plan |
| Test specification document | **Test specification** **source** is test description written manually in form of Doxygen style comment in each test case or test scenario, ideally before coding is started. Includes also common XML articles and customized templates, specific for each release.  Test specification source template constitutes ,   * + - * 1. test case/suite description added in C source code with doxygen parser in the test script         2. custom xml files for test specification table generation like chapter Revision history, Test Environment and Constraint, Test Environment, Known limitations and assumptions, Acronyms and Definitions, Driver specific variable   **Test specification document** is automatically generated using DOCATO tool where the various input files which are listed under Test specification source is fetched to create the Driver specific test specification (PDF format) document |
| Test source files: test cases, test suites | **Test cases** are written using standard template.  **Test suites** are also written manually – the easiest way is to derive from any existing test suite. Test suite shall contain primarily all initialization common for included test cases and registration of these test cases.  Test case source file constitutes,  a. test case/suite script in C language  b. test case/suite unique traceability tags added |
| Test logs | Test log is generated by test framework for each test suite after execution time. This is input to test reports. |
| Test reports | Test reports are generated by automated test framework. Additional scripts included in the build environment are used to reformat and join individual test reports.  Test report template file constitutes,   * + - * 1. Autogenerated Driver summary report         2. Compiler warning reports |
| Traceability matrix | Traceability matrix is generated using ReqTracer tool, which uses inputs from Doors (requirement database), Enterprise Architect driver design files, source code and test source code. Dedicated script is used prepare index of test cases for ReqTracer to speed up preparation of final traceability matrix.  Reqtracer tool generates,  Test case and requirements traceability & analysis mapping |
| Code coverage reports | LDRA tool generates,  MCDC, statement & decision coverage  Detailed code coverage report |
| MISRA reports | Coverity tool generates,  MISRA violation report |
| Cyclomatic Complexity reports | Coverity tool generates,  Cyclometric complexity & nesting depth |
| Profile report | Trace 32 macro tools measure the performance and generates,   * + - * 1. Profile report with MIN, MAX and AVERAGE execution time |
| Code Size report | RAM Size tool after building all the tests generates,   * + - * 1. RAM size report for min, max, normal scenarios         2. Code and stack estimation |

Table 5 SmartTRX deliverable Test artefacts

# Acceptance Reviews and Approvals

* + - * 1. Refer link for “Automotive Verification Procedure - BCaM 7” document details the reviewer and approvals role for the test artefacts documented in BL-RFP QMS page:

<https://nxp1.sharepoint.com/teams/25_8/VandV/Verification.aspx>

* + - * 1. Refer link for “BU AUTO” review checklist which shall be used for test artefacts reviews

<https://www.collabnet.nxp.com/sf/go/doc427148?nav=1&pagenum=1&pagesize=15>

* 1. Refer collabnet link for test artefacts Inspection record

<https://www.collabnet.nxp.com/ctf/documents/list/projects.smarttrx/docman.root.es0.260_review_files.sw.sw_integration_and_validation>

# Annexes

## 9.1 List of Tables

[Table 1 SmartTRX software subsystem 4](#_Toc41402992)

[Table 2 Unit level Test type mapping to SW subsystem 18](#_Toc41402993)

[Table 3 Integration level Test type mapping to SW subsystem 22](#_Toc41402994)

[Table 4 Qualification level Test type mapping to SW subsystem 24](#_Toc41402995)

[Table 5 SmartTRX deliverable Test artefacts 28](#_Toc41402996)

## 9.2 Test Type description

#### Requirements Based Testing

Analysis of requirements leads toprecondition of the test and sequence of steps to be performed by the component. Test vectors definition is not in scope of this technique, typically it is combined with error guessing test technique to select proper test vectors, but error guessing method does not ensure meaningful and measurable coverage of space of test vectors.

#### Equivalence Classes

Equivalence class is the partitioning of the input variable’s value set into classes using some equivalence relation. Equivalence Class Partitioning is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed.

#### Boundary Testing

Boundary testing is subset of equivalence classes testing, focused on test vector on the border between functional and non-functional equivalence classes. Test preparation starts also with analysis of all API parameters, but then tests are implemented manually, without the automated generator based on pairwise algorithm. Boundary test shall be written for API parameter based on integer type, for enumeration types boundary test are focused primarily on channel parameter, representing range of available resources. To test this, range a configuration with all available channels is created and then correctness of the first and the last channel is verified.

This test technique is suitable to create test vectors for component testing**.**

#### Static Analysis

The source code that is part of the build process to obtain STRX One-chip variant release candidates shall be statically verified using Coverity central analysis, with checkers enabled per Coding Style. The false positives and accepted violations will be marked as such in Coverity and will not be reported during the next analysis.

Desktop analysis may be used by developers, e.g. for pre-commit verification.

#### Fault Injection

Fault-injection tests involve the deliberate introduction of faults to test system robustness and error-handling capabilities. Faults can be introduced directly into the code (compile-time injection) or using software triggers that cause specific scenarios to occur in a running system (runtime injection). It ensures that the system can handle and recover from fault or error conditions and identifies design weaknesses where a single fault could potentially be propagated into a severe error or systemic failure.

#### Error Guessing

Error guessing isa test technique in which test cases used to find bugs in programs are established based on experience in prior testing. The scope of test cases usually relies on the software tester involved, who uses past experience and intuition to determine what situations commonly cause software failure or may cause errors to appear.

#### Performance Test

Testing conducted to evaluate the compliance of a system or component with specified performance requirements. Performance attributes such as 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 busses) are measured

## Test Specifications and Traceability

Test specifications and traceability are re-used from BL-AP for STRX software content that does not have differences in requirements with respect to S32R41. For software content which has differences in requirements compared to S32R41 and for software content which is present only in STRX, the test specifications and traceability shall be created based on new or changed requirements.

|  |  |
| --- | --- |
| **Test Item** | **Test Specifications and Traceability** |
| Application Bootloader | Reuse from BL-AP |
| Platform SDK | Reuse from BL-AP |
| Safety SDK | Reuse from BL-AP |
| Radar SDK | Reuse from BL-AP |
| RFE Software | To be created by PPA |

# Document Information

## References

| ***Item*** | ***Description*** |
| --- | --- |
| ISO 26262 – Part 6 | https://www.iso.org/standard/68388.html |
| BL-RFP SW Quality Policy | [Collabnet doc356947​​​](https://www.collabnet.nxp.com/sf/docman/do/downloadDocument/projects.blida_subqms_ccb/docman.root.bl_ida_qms_published.software_design.processes/doc356947) |
| BU-Automotive SW Procedure | [NXPOMS-1719007347-3853](https://nxp1.sharepoint.com/sites/OMS/_layouts/15/WopiFrame.aspx?sourcedoc=%7b18C22CC3-44F0-45E0-BAF3-63BDD2F0DDD7%7d&file=BU%20AUTO%20Software%20Procedure%20-%20BCaM7.0.docx&action=default&DefaultItemOpen=1) |
| Automotive SPICE 3.1 | <http://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf> |
| Software Configuration Management Plan [SCMP] | <https://www.collabnet.nxp.com/sf/go/doc405541> |
| Software Project Management Plan[040.20\_SmartTRX\_SW\_PMP] | <https://www.collabnet.nxp.com/sf/go/doc430900> |
| Software Test activity | <https://www.collabnet.nxp.com/sf/go/doc417653> |
| Software requirement specification (RFE,RADAR) | <https://doorsng.nxp.com/rm/web#action=com.ibm.rdm.web.pages.showFoundationProjectDashboard&componentURI=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Frm-projects%2F_ZWprQKXvEeicDL3MQQBumw%2Fcomponents%2F_ZaUDQKXvEeicDL3MQQBumw> |
| Software requirement specification (IPCF) | [https://doorsng.nxp.com/rm/web#action=com.ibm.rdm.web.pages.showProjectDashboard&componentURI=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Frm-projects%2F\_woakoWyKEeeut8Jrs2apBA%2Fcomponents%2F\_wq4CwGyKEeeut8Jrs2apBA&vvc.configuration=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Fcm%2Fstream%2F\_wq\_-kGyKEeeut8Jrs2apBA](https://doorsng.nxp.com/rm/web#action=com.ibm.rdm.web.pages.showProjectDashboard&componentURI=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Frm-projects%2F_woakoWyKEeeut8Jrs2apBA%2Fcomponents%2F_wq4CwGyKEeeut8Jrs2apBA&vvc.configuration=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Fcm%2Fstream%2F_wq_-kGyK) |
| Software requirement specification (Real Time drivers) | <https://doorsng.nxp.com/rm/web#action=com.ibm.rdm.web.pages.showFoundationProjectDashboard&componentURI=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Frm-projects%2F_iScLEfWNEeaiUPR28B-GcQ%2Fcomponents%2F_iTyO4PWNEeaiUPR28B-GcQ> |
| SmartTRX Software Architecture specification | <https://www.collabnet.nxp.com/sf/go/doc347357?nav=1&pagenum=1&pagesize=15> |
| BL RFP QMS PUBLISHED Software artefact templates | <https://nxp1.sharepoint.com/teams/25_8/Software/SW%20Design.aspx> |
| [SW Quality Assurance Plan](https://www.collabnet.nxp.com/sf/docman/do/downloadDocument/projects.smarttrx/docman.root.es0.070_project_quality_assurance/doc432165) | <https://www.collabnet.nxp.com/sf/go/doc432165?nav=1&pagenum=1&pagesize=15> |

## Terms/Acronyms and Definitions

### **10.2.1 Standard Terminology**

|  |  |  |
| --- | --- | --- |
| **Term** | **Origin** | **Description** |
| Validation | ISO/IEC/IEEE 29119 | Validation demonstrates that the work item can be used by the users for their specific tasks. |
| Verification | ISO/IEC/IEEE 29119 | Verification is confirmation, through the provision of objective evidence, that specified requirements have been fulfilled in a given work item. |
| Functional requirement | IEEE Std. 610-1990 | A requirement that specifies a function that a component or system must perform. |
| Non-functional requirement | IEEE Std. 610-1990 | A requirement that does not relate to functionality, but to attributes such as reliability, efficiency, usability, maintainability and portability |
| System | Automotive SPICE V3.1 | A collection of interacting items organized to accomplish a specific function or set of functions within a specific environment. |
| System test | ISO/IEC/IEEE 24765 | Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. |
| Integration | Automotive SPICE V3.1 | A process of combining items to larger items up to an overall system. |
| Integrated system item | Automotive SPICE V3.1 | A set of items that are integrated into a larger assembly for the purpose of integration testing. |
| Integrated software item | Automotive SPICE V3.1 | A set of software units or items that are integrated into a larger assembly for the purpose of integration testing. |
| Integration testing | Automotive SPICE V3.1 | Testing in which items (software items, hardware items, or system items) are combined and tested to evaluate the interaction among them. |
| Functional testing | ISO/IEC/IEEE 24765 | Testing conducted to evaluate the compliance of a system or component with specified functional requirements. |
| Software Unit | Automotive SPICE V3.1 | Part of a software component which is not further subdivided |
| component | IEEE Std 610.12-1990 | One of the parts that make up  a system. A component may be hardware or software and may be subdivided into other  components. |
| Code review | Automotive SPICE V3.1 | A check of the code by one or more qualified persons to determine its suitability for its intended use and identify discrepancies from specifications and standards. |
| Black-box testing | Automotive SPICE V3.1 | Method of requirement testing where tests are developed without knowledge of the internal structure and mechanisms of the tested item. |
| Baseline | ISO/IEC/IEEE 24765 | A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and can be changed only through formal change control procedures. |
| White-box testing | Automotive SPICE V3.1 | Method of testing where tests are developed based on the knowledge of the internal structure and mechanisms of the tested item. |
| DEM | AUTOSAR | Diagnostic Event Manager handles and stores the events detected by diagnostic monitors in both Software components (SW-Cs) and Basic software (BSW) |
| DET | AUTOSAR | Development Error Tracer to support error detection and tracing of errors during the development of Software components and other Basic Software Modules |

**ISO/IEC/IEEE 29119**: Software *and systems engineering [Software testing]*

**ISO/IEC/IEEE 24765**: Systems and software engineering

**IEEE Std. 610 1990**: IEEE Standard Computer Dictionary: A Compilation of IEEE Standard

Computer Glossaries

**Automotive SPICE 3.1**: **A**utomotive **S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination

### **10.2.2 Terms/Acronyms and Definitions**

| ***Acronym / Terms*** | ***Definition*** |
| --- | --- |
| MCAL | Micro Controller Abstraction Layer |
| CDD | Complex Device Drivers |
| AUTOSAR | AUTomotive Open System ARchitecture |
| IPCF | Inter Processor Communication Framework |
| HSE | Hardware Security Engine |
| SDK | Software Development Kit |
| PMP | Project Management Plan |
| RFE | Radar Front End |
| SW RS | Software requirement specification |
| SW AS | Software Architecture specification |
| PCA | Project Concept Approval(phase gate) |
| PDA | Project Definition Approval(phase gate) |
| PPA | Project Planning Approval(phase gate) |
| SCMP | Software configuration management plan |

## Revision History

| Document Author | Version | Date | Description of Change |
| --- | --- | --- | --- |
| Nanda Kumar G | V4.6 | 06 Jul 2020 | * + - 1. Table 4 unwanted entry removed       2. Section 10.4 Link to SW quality assurance plan       3. Section 3.6 inclusion of SW quality goal       4. Section 3.7 extension of schedule description       5. Section 3.6 test types are aligned with test types detailed in section 4.1       6. Section 3.6 RFP step-2 added more detail       7. Section 4.1.2.1.5/.6 inputs section are added       8. Section 4.1 all the testing types inputs are improved |
| Nanda Kumar G | V4.5 | 29 May 2020 | Review comments fixes  refer review version 12 |
| Nanda Kumar G | V4.4 | 21 May 2020 | Software architects & quality team comments updated |
| Nanda Kumar G | V4.3 | 23 April 2020 | Internal review comments updated |
| Biju Ravindran | V4.2 | 27 March 2020 | Added test specifications and traceability sections |
| Nanda Kumar G | V4.1 | 09 Mar 2020 | Software/Test architect review comment fixed.  Baselined version |
| Nanda Kumar G | V4.0 | 26 Feb 2020 | PCA test plan scope section1 to section3 are aligned with software architects |
| Nanda Kumar G | V3.1 | 06 Feb 2020 | Software Architects Review comment updated refer review version 6 and 7 |
| Nanda Kumar G | V3.0 | 21 Jan 2020 | Software architect review comment updated refer review version 5 |
| Nanda Kumar G | V2.1 | 20 Nov 2019 | Software testing activities are detailed with better illustrations |
| Nanda Kumar G | V2.0 | 19 Nov 2019 | Review comments on overall validation from SW architect are fixed |
| Nanda Kumar G | V1.2 | 18 Nov 2019 | RADAR SDK review comments are fixed |
| Nanda Kumar G | V1.1 | 11 Nov 2019 | Correction to SDKs present in RFE validation software, driver and application software |
| Nanda Kumar G | V1.0 | 8 Nov 2019 | Initial version SmartTRX SW Test plan |