Software Test Plan Template Guide

Michael Stahl
Intel

EuroSTAR Huddle

eBook

EuroSTAR
Huddle

Welcome

At EuroSTAR, our core purpose is to help software test professionals to achieve their absolute full professional potential and to inspire them through community and collaboration to help each other.

From the EuroSTAR Huddle for testers wishing to learn and improve to the annual EuroSTAR Conference, we have been bringing testing and quality assurance professionals together since 1993.

We are delighted to present this Software Test Plan Template Guide and the accompanying template written by Michael Stahl, who has previously spoken at our EuroSTAR Conference.

Enjoy!

The EuroSTAR Team

Michael Stahl

Michael Stahl is a SW Validation Architect at Intel. In the last 20 years Michael tested code for Smart TVs, graphics cards, computer-vision applications based on 3D cameras and Intel's Active Management technology and Security Engine.

In his role, Michael defines testing strategies and work methodologies for test teams and sometimes even gets to test something himself - which he enjoys most.

Michael presented papers in SIGiST Israel, STARWest, EuroSTAR and other international conferences and is teaching SW Testing in the Hebrew University in Jerusalem.

You can view his papers and presentations at www.testprincipia.com

Template Download

This ebook is a guide to a Software Test Plan Template. Click the button to download the editable template.

DOWNLOAD FILE HERE

Software Test Plan Template Guide

Table of Contents

Revision History

Date	Revision	Author	Summary of Change
29/Nov/2021	1.0	Michael Stahl	eBook version

Preface

A bit of history

The content of this eBook was written while I working as a SW Validation Architect at Intel. At the time (2017) I was leading the effort to make our product comply with ISO 26262 (Functional Safety standard - FUSA). I made much use of work I did earlier in defining Software Test Plan templates and in teaching

Test Planning. One of the things I always struggled with was the template length. Just dumping the template on test engineers and asking them to use it proved impractical. Testers did not know what to do with this template. So I added explanations for each section. But then the resulting template was tens of pages long, and the sheer length of the document scared testers and caused them to lose any appetite they may have had to engage in writing a test plan.

With FUSA, things became even worse. The standard mandates writing Test Plans, for each test level. So now we had to have three templates (unit, integration, system), each rather large with explanations. Not only that: the explanation text was pretty similar in many places, so each improvement of the explanations had to be done three times. A mess.

The solution was to create two files: One would hold just the templates sections (with bare-minimum, one-liner explanation for each section) and a Guide, where sections are explained in more detail. What you have in this eBook is the result of this approach: A Guide document and a Template document.

The template part contains sections for all three test levels. A correct use of the template is to remove the sections that don't apply, so you end up with just the sections you need for the selected level.

I admit it is still rather long. Use this is a starting point for YOUR Software Test Plan template. Remove anything you feel is redundant and add whatever I missed. I do recommend reading the Guide text for each section you decide to remove and consider if removing it is a good idea. There may also be cases where you feel that certain information is important but fits better in a different place in the template.

Acknowledgements

The first version of the Test Plan template was created as a group effort in the WiFi organization at Intel.

The test plan and guide presented here were written by me. It was thoroughly reviewed by several Intel people, many of whom have a much better knowledge and experience with functional safety than me. The comments and contributions made by the reviewers made the template and guide significantly better. It's my duty and pleasure to thank the following contributors: Cosmin Munteanu; Giovanni Sartori; Gloria Wirth; Luca Fogli and Linda Zavaleta.

In some cases, I use terms or material adopted from ideas of other people. Things I read or heard in conferences. I added the source in those places.

Disclaimers

This eBook and the accompanying template are presented as-is: A best-effort attempt to help SW testers with test plans development. It does not guarantee that the resulting test plan would be any good. That part is up to the user.

Feel free to change the template to suit your situation and organization.

My experience with Functional Safety and ISO 26262 is already three-four years old (I worked on Functional Safety in 2017-2018). I had no experience with IEC 61508, and any reference to it is courtesy

of the contributors. I did not invest time while editing the template and guide for this eBook publication to ensure it is up to date with any (possible) changes introduced since 2017. So be extra careful when applying this template to Functional Safety project. Use this material here is a starting point and check for any changes that may have taken place in ISO 26262 IEC 61508 since 2017.

Michael Stahl

Nov'2021

Mail: michael.m.stahl@gmail.com

LinkedIn: https://www.linkedin.com/in/michaelmstahl

Web: http://www.testprincipia.com

General

How to use this guide

The guide is rather long. It's a reference material which you are not expected to read in one seating. Use this reference to learn what information is expected in each section of the SW Test Plan template.

Do make sure to read all the General section (the section you are reading now). If you never wrote a test plan or never used the SW Test Plan template, not reading this part almost guarantees you will lose more time than you saved – not to mention the frustration when you find out you did not do it right. Trust me on this one.

What is a Test Plan?

A Test Plan is a detailed description of a test activity. It documents what is tested, how, why and the means for doing it.

The most important part of the Test Plan are the details of the test strategy:

- How you test the item-under-test (e.g. what test techniques you use)
- Why this way is adequate to achieve an acceptable test coverage. There are unlimited numbers of tests you can think of; the test plan explains which tests you selected out of this unlimited number and why you think your selection is a good one.
- What configurations (platform and OS) you cover
- What tests run in different types of test cycles (e.g. BAT, regression, CI, milestones)

Additionally, it contains logistics and environment information: details about the test setup and environment; about test automation and tools you use and about the resources involved.

Reading the test plan gives the reader a good understanding on how you tackle the test task that the plan covers.

A project usually has more than one Test Plan. First, it may have a different test plan for different test levels (unit, Integration and system tests; for some processes – such as functional safety – these test plans are required). For each test level you may have more than on test plan document. For example, at system test level, there could be Master Test Plan (also known as a Project Test Plan) that encompasses all testing activities on the project; further detail of particular test activities could be defined in one or more test sub-process plans (i.e. a performance test plan) or in feature-specific test plans (Feature Test Plan).

A Test Plan is a Word document. It is not a list of test cases. There is of course a connection: your test cases **implement** the test strategy outlined in the Test Plan.

Guide structure

In this document, the SW Test Plan template is referred to as the Template; the text you are reading now is referred to as the Guide.

The Guide covers all the sections of the SW Test Plan template. To avoid confusion between section numbers in the Guide and section numbers in the Template, the Guide does not use numbered sections, but use the same section titles.

The following lists where Template sections are covered in the Guide.

- General information about the Template and about writing test plans is in the General section.
- The template's front page, Revision and Table of Content sections are explained in the "Front Matter" section.
- The first two sections in the Template ("Introduction" and "Document Scope") as well as sections common to all test plan types ("Risk analysis" and "Schedule, Project management and Staffing") are explained in similarly called sections.
- The "Unit Test Plan", "Integration Test Plan" and "Master | Feature Test Plan" sections are explained in one section of this guide – "Software Test Plan sections" – to avoid having to repeat similar explanation for each test level. When the explanation differs between test levels, it is clearly marked so; otherwise, the explanation is similar for all test levels.

Color code used in this Guide:

- Pink: general instructions that apply to all test plans – safety related features or otherwise.
- Green: instructions that apply to documents dealing with safety-relevant features
- Black (regular): Preface text

- *Black (italics): Text that can be put in the test plan as-is or with slight modifications.*

Why write a Test Plan?

Developing high quality software products is a non-trivial effort. Research shows that proper documentation helps in achieving this. Additionally, if your product is for the automotive and industrial sector, international standards mandate a strict regime of documentation, of which this test plan template is a part.

While it seems the main reason you need to write a test plan is to make the process happy, writing a test plan has real benefits for you (the test engineer) and for other stakeholders, such as architects, developers and project managers. The act of writing a test plan – putting ideas and thoughts on paper about how to test something – helps you understand better what you need to do and what preparations to make so that you have a chance of meeting your commitments. Having a clear document describing what the test engineer plans to do aligns the stakeholders' expectations of "what is it we get from the test people". It improves the chances of getting meaningful feedback from the stakeholders and of finding test gaps or non-intentional overlaps with other test activities.

If you do it "after the fact" – when you are already deep into the test project, the benefit is that it forces you to review what you are doing and that it creates a readable documentation for new test engineers. If you do it very early in the project (way before Pre-Alpha is best), you gain much more. It will force you to think about all the details you need to plan for. It will help you get through the project in one piece.

In short: Just Do It. You can thank me later.

Terminology

Text books talk about three main software test levels: Unit, Integration and System test. The Template is designed based on my position regarding what each of these test levels means. This may not be the same as your view on the subject or how your team refers to different test levels. To learn more about the definition of Unit, Integration and System test that was used when creating the Template, see Appendix A – Test Levels. Based on this, you can decide what part of the Template fits your activities.

Template structure

The Template is designed to be used for a number of test plan documents:

- Unit Test Plan
- Integration Test Plan
- Feature Test Plan
- Master Test Plan

The test plan template is designed according to ISO 29119 (2013) definition of a Test Plan and as such satisfies the requirements of ASPICE. To learn more how this templates relates to relevant standards, see Appendix B – Relevant International Standards.

Depending on your organization, you may have separate documents for each test level or a combined document for all test levels. A test plan document may cover a single feature, a set of features or even the whole product.

Unless you intend to have one test plan document for all three levels (unit, integration and feature), you need to remove from the Template the sections that don't apply to the test plan you write. For example, if you write a Unit Test Plan, remove sections 4 and 5 from the Template, so you are only left with the Unit Test Plan section and the general-purpose sections.

When using the Template for a Master Test Plan, remove sections 3.0 and 4.0 (unit and integration tests plans). Section 5.0 (Master Test Plan) will now become section 3.0, and this is your Master Test Plan template.

When using the Template for… OK; you got the idea.

Where to start?

To avoid running out of steam before you get to the most important sections, start filling the Template sections in the following order:

1) **Test Scope**
   i. Unit and Integration Test Plans: **Test Items** sub-section
   ii. Master and Feature Test Plans: **Features to be tested** sub-section
2) **Test Approach: A**ll the sub-sections – but start with the **Test Design Specification** section
3) **Test setups** & **Hardware & Lab** sub-sections in the **Test Environment** section
4) **Test Scope:** Complete the rest of the sub-sections
5) Do the rest of the document in any order you feel like

Make sure to hold a review of the plan. You will be surprised at some of the things you did not think about and someone else did. Don't wait until you wrote the whole test plan before asking people to review it, especially if this is the first time you write one. Follow the recommended section-writing order above. Once you have 2-3 items in the Test Design Specification sub-section of Test Approach, ask someone in your team that has experience in writing a test plan to review your document. Then, when the document is ready (version 0.3 and 0.5 - see versioning standard in the Revision History section) – do a review with the appropriate audience.

Technical notes

- This template is a memory aid and a checklist. It covers a lot of aspects that are relevant, in some projects, to testing. It includes sections required by ISO 29119 and ISO 26262. This means it includes sections that you may see as redundant in the context of the code you are testing. This is fine. Just write "NA" and move on with your life.
- In some cases the template requests information that in your organization is available in other places. There is no reason to repeat it in the test plan document. Reference those other places instead of duplicating the details.
- Don't delete sections. Write NA. This will tell the readers that you thought about the topic of the section and made a decision that this section is irrelevant. They can then agree… or not. It will give your reviewers a chance to tell you about something you missed (I can already hear them: "Oh! But this is VERY applicable!"). The template is also a memory aid for your reviewers!

  There are two exceptions to this rule:

  - It's recommended to remove test-levels sections that are not covered in your test plan.
  - If the help text in the Template specifically allows deletion of sections if they are not applicable, you are OK to delete them
  - In some places, there are cross-references in the Template. If you removed sections, the references may be wrong and some of them lead to a section that was deleted. Fix that before releasing the document. And refresh the Table of Contents…

General writing tips

The following are correct when writing most documents – not only a SW Test Plan.

- If you are writing the test plan before there is code, you will naturally use future tense to describe planned actions ("for feature x we will use this and that test technique"). However, the test plan continued to live after the product is coded; possibly even after the product is shipped. At that time, it's kind of strange to read what you "will" do. So: use present tense. For example: actions ("for feature x we use this and that test technique"). Even if you are talking about something you have not yet done.
- Whenever using dates, specify the month in letters, not a number (e.g. 03/Jan/2018 and not 03/01/2018). This will avoid misinterpretation between US date notation and the natural, logical and clear notation used by us normal Europeans.
- When providing a URL, it is better to give the link to the folder or web site where the document is and not a direct link to the document; this will save you from losing a link due to a filename change.

Front Matter

Table of Contents

This is just a standard "table of contents" of Word. The only thing to remember about it is to update it whenever you update the test plan.

- Select all text (control-A)
- Hit F9; for any popup, select "Update entire table". You may get a number of these, depending on the extent of the editing done

If you use any cross-reference within the document, the references will be updated as well as a result of this action. Which is a good thing.

Revision History

Follow your company or org standard version taxonomy.

Here is one that I found useful:

Revision	Description
0.1	Copy of the empty template, with a file name change. Nothing to show anyone yet.
0.3	Contains items 1-4 from the "Where to start" recommendation; enough content to scope effort; direction identified; stakeholders and review schedule identified.
0.4	Ready for peer review
0.5	Incorporated feedback from peer review; direction confirmed, contract with other groups; can start engineering execution
0.6	Ready for stakeholder review: all relevant sections filled in and content complete
0.7	Incorporated feedback from stakeholder review; resend for ratification
0.8	Ratified: change control started; doc baselined
1.0	Approved changes to 0.8 baseline; includes customer feedback
1.n	Final "as implemented" update

Revision	Date	Author	Description

Opens

List issues that are open at the time of publishing the test plan. In theory, there should not be any opens after v0.8. In real life – for non-safety relevant items - there are. When that happens, either turn them into an action-required list, or just keep the Opens list here and make a note how you will work around this open until it is resolved.

For Safety-relevant features or code, you really MUST close all opens before your test plan is approved for version 0.8.

Issue	Description	Owner	Status

Introduction

This is the first text the reader will see that is actually related directly to the test subject. So this is the place to write an overview about the product or feature that your test plan covers. Depending on the test level, you may put here different information.

The goal of this section is that the reader, once done reading it, will know what the test plan is addressing and have a good context to be able to follow the rest of the document. If it's an Integration Test Plan – give an idea of what is being tested; if a Feature – talk a bit about the functions provided by the feature (e.g. What does it do? How is it used?). Etc.

Other information that is relevant here:

- Uniquely identify the product being addressed in this document by product name or code name. Include the version number of the product if available and relevant.
- Provide the name of the organization that owns this document.
- For FuSa – make sure to mention the ASIL(s) assigned to the SW elements covered by this test plan. For cases where there are different levels of ASIL for different parts of the SW covered by this test plan, make sure to be very clear about what ASIL level is each component. An alternative place to do so is in the "In Scope" sections. See further comments there.

Whatever you write here, it should be concise. This section should not be longer than 1 page; ½ a page or even just a few sentences is more like it.

- When describing something that is already covered by another document (for example, a feature may be described in detail by an Architecture document), reference that document.

Even if you do so, it is highly recommended you not ONLY reference the other document but do give a short overview here. Just making a reference means that to get context the reader needs to first read the reference document before continuing. This is a distraction. In many cases it means getting permissions and reading a long document with a lot of details not needed for understanding the test plan. It is much more useful to have here a short overview with enough details to give the reader a general context; reference the more detailed source to allow interested readers to dig further.

Purpose

Briefly describe the purpose of this document. The test of what component does it describe? What test levels are covered? What are the goals of this document? In many cases, you can just use the text below, filling in the missing information (read it before using it. Make sure it's OK for you!).

This Software Test Plan is used as a tool to create a coherent and well-coordinated software <Master | Feature | system | Integration | Unit | Component| … > test strategy of the <component> feature-set by <team name>.

When done reading this document, you will have a fairly good idea how <team> plans to conduct <test level> testing of the <component> of <this product>, on <OS, platforms>.

If this document is part of a series of documents that are related to each other, consider mentioning this. If this document is (in part) created to comply with a standard, consider mentioning that too.

Example (for relating to a series of documents and to a standard):

This test plan is a part of a series of documents that support the ISO 26262 requirements for the Mechaton-AD SoC used as a Safety Element out of Context (SEooC) for the Advanced Driving Assistance Systems (ADAS).

Audience

This section is a requirement of ASPICE. It must be in the test plan document even if you think it does not add much benefit (well, it doesn't, does it… but we all want ASPICE to be happy, right?). In most cases, the text below is generic enough (and correct enough) to be used as is. Note the addition of safety auditors for safety-relevant code.

The audience for this document are architects, developers, testers, project managers as well as safety auditors.

Add other audiences if that's how your org works.

Acronyms and Terminology

Provide definitions for acronyms and terminology used in this document. Use only those which are in this specific document; do not include any redundant terms – this is not a replacement for Wikipedia.

Sort alphabetically.

If you have a document that contains all the terms and acronyms, you can just reference that document. While a possible and acceptable approach, it does mean that the reader of THIS document will need to consult another document to get terms and abbreviations explained. Quite annoying. Decide what makes sense in your case and how much you want to annoy the reader.

It is OK to assume a specific term is so well known that it does not need to be listed here ("OK" is a good example). However, if any of your reviewers ask to add it in – don't argue. Just add it in.

As an example, the table below includes acronyms and terminology used in the Guide.

A summary of terminology used in this document is outlined in the table below.

Table 1. Acronyms and Terminology

Term	Description
ASIL	Automotive Safety Integrity Level. One of four levels to specify the item's or element's necessary requirements of ISO 26262 and safety measures to apply for avoiding an unreasonable residual risk, with D representing the most stringent and A the least stringent level (ISO26262-1).
BAT	Build Acceptance Tests; also known as “Smoke Tests”
CI	Continuous Integration
FuSa	Functional Safety.
FTP	Feature Test Plan
MTP	Master Test Plan
OS	Operating system
OTS	Organizational Test Strategy. A document explaining “how our team does testing”. Proposed by ISO 29119 but not mandated by ASPICE or ISO 26262. See more in “Reference Documents”.
SR	Safety relevant
Test Level	Unit test, Integration test, System test are all “test levels”.
Test Module	See explanation in “Features to be Tested” section

Reference Documents

Provide a list of documents which are referenced in this document or were used as input when creating this test plan. Identify where each document can be found and the revision used.

The Mnemonic is a short-hand for the document name. It can be used within the test plan to identify a specific reference without having to write its full name.

One document that is worth referencing if you have it, is the Organizational Test Strategy (OTS). See Appendix B – Relevant International Standards where the OTS is explained in more details. Whenever the activities you do for this test plan follow the generic processes of your organization, you can reference the OTS and save re-documenting things.

ISO 26262-6:2018, sections 9.3.1 and 10.3.1 list the pre-requisite information that should be available when writing a test plan. Consider referencing here the pre-requisite documents you actually used when developing this test plan. Not a must.

The entries in this table are examples of documents you may want or need to reference. Delete / replace as needed and add your own references.

Table 2 Reference Documents

Mnemonic	Document	Document Location
[SAS]	Software Architecture Specification	http://....
[MTP]	Master Test Plan	http://....
[PRD]	Product Requirements Document	http://....
[OTS]	Organization Test Strategy	http://....
[REQ]	Requirements document (or URL to database)	http://....
[ChMgt]	Change management process / strategy document	http://....
[BugMgt]	Bug management process	http://...
[C&C]	Testing of ISO 26262 Configurable Software	http://testprincipia.com/presentations-and-papers-on-software-testing/#heading12

Document scope

Section 2.0 in the Template was added specifically for proper FuSa documentation. It lists the relevant sections in ISO 26262 and IEC 61508 whose requirements are addressed by the test plan.

Prerequisite documents
In Scope
Out of Scope

For features that implement safety relevant (SR) requirements, just leave all these sections with the information already provided in the Template.

For features that don't implement SR requirements, delete the contents of each sub-section and write there "NA" instead.

Software Test Plan sections

Test Scope

This section provides information about what is covered – and what is not – by the test plan. Accurate information here will avoid misalignment between what the validation team commits to cover and what other teams assume the team covers. It also allows reviewers to note if test activities committed here are also committed by other teams. In these cases, duplication of effort may be avoided – or at least be done consciously.

Specifically for Integration Test Plan, explain here what Integration Tests mean in your organization (See discussion in "Terminology").

Safety requirements

If the code under test implements no safety requirements, write here "NA".

This section is mandatory to fill in when the code does implement safety requirements, for which this test plan will be assessed according to 26262.

List, or give reference to the list of the safety requirements implemented by the code under test. Later in the test plan (in the "Test Strategy" section), you will discuss if and how these requirements get special attention.

If your requirements are managed in a database (I sure hope they are) – give a URL to the DB and how to get the requirement list covered by this test plan (e.g. give the URL to a query).

For ISO 26262 compliance, ensure that this section (together possibly with the "Test Items" section) covers the following:

- What are the work products and/or safety integrity requirements to be verified? [ISO 26262-8:2018; 9.4.1.1a]

Test items

Unit test:

Enumerate the code units that are in scope of this test plan. These would be the code files that are tested. Usually, since we don't want to have here a long list of files, just give a general name, and a URL to the relevant folder in your code repository.

For Safety related code, add also the path to the Safety Plan location.

Example:

Table 3 Unit Test Items

ID	Unit	Path
Item.1	XYZ app	//…
Item.2	ABC library	//…

Integration test:

Enumerate test items (executable binaries) of the product that are in scope of this test plan.

Test items include the pieces of the software whose integration with one another (or with the system) is the target of this integration test plan.

Examples depend on how you define "Integration test":

- Two or more separate modules of a large system
- A library and the code that use it
- A new feature added to an existing program module
- Code whose interaction with the HW is the goal of this test plan
- Code whose interaction with the OS is the goal of this test plan
- Code that is tested by the CI system

Integration tests need to cover the following (you can opt to cover some or all of these in the Feature Test Plan):

a) Show that the code does what was specified in the software architectural design

b) The hardware-software interface (unless done by another team. E.g. the HW team; if so, state it in the "Test Scope" section).

c) Verify that non-functional requirements are met

d) Verify correct behavior in case of erroneous inputs

e) Verify the product has sufficient resources to support the functionality

f) Verify the implemented safety measures

Example (when your definition of integration is "interfaces testing")

Table 4: Integration Test Items

Test Item	Interfaces
Services.dll	dll - OS APIs
	dll - application level APIs
	dll – remote authentication server REST Messages
hid.sys	SW – HW (via shared memory registers)
	OS – driver (using IOCTLs)

Example (when your definition of integration is "functionality of a part of the code")

Test Item	Test targets
Services.dll	Interfaces with the OS, application and remote server
	Basic functionality
	Specific performance aspects

Feature

Test items include all pieces of the software that make up the product (e.g. software binaries, dll files, configuration files). If the product is provided to the customer as sources, the source files and any related files such as build scripts or make-files should be included in the inventory of test items. User documentation and the installation package (if exists) should also be listed.

Sometimes the binaries under test are those that get installed by the software's installer. In this case, there is little benefit in listing all the binaries here and you can just list the installer and mention that the Test Items are the resulting installed files.

Specify version numbers if available. Where applicable, specify the "main focus area" that will be tested in each Test Item in the context of this test plan.

Example:

Table 5: Master|Feature Test Items

Test Item	Description & version	Main focus area
ABC DLL	Authentication and Access control engine	Authentication
XYZ Driver	Device driver (User mode)	Authentication support

Features to be tested

This section appears only for Master | Feature test plan.

This is a breakdown of a high level feature (or: "component") into "Test Modules". The test module names and the details of the teams responsible to test them are listed in a table. In some cases, a single test module covers a number of aspect and these are listed as "Additional breakdown". Later in this document (in Test Design Specification) you will add more details about each test module – e.g. the test strategy.

Teams can decide that a certain Test Module must exist in each FTP (e.g.: decide that a "Security" test module must exist in each test plan). In such case, you may want to create a local version of the template for your organization and already have it populated with the test modules you mandate.

Each test module focuses on a specific aspect of the component (a feature or a capability). As a goal, a test module should be:

- Well differentiated and clear in scope (*)
- Balanced in size and amount of testing (*)

(*) Not always possible, but a good goal

Note: The term "Test Module" and its definition above was adopted from Hans Buwalda.

Add test modules to cover special test areas or test types that are not feature-specific (e.g. installation).

Add test modules for non-functional aspects (performance, power, CPU/GPU utilization, security testing, load, stress, reliability, MTBF etc.)

Add test modules for any certification, compliance requirement or regulatory requirements (e.g. Microsoft's tests (known as WHQL); Bluetooth certification).

If the feature is tested in pre-silicon stages, add a pre-silicon test module where you can explain the setup and testing done in pre-silicon stage.

Add or remove columns with additional data as fits your case. The important part is that the list includes all the "moving parts" of the component and that if something is NOT in this table it means you are not planning to test it.

If the table of features becomes unwieldy (too wide to manage in Word), consider embedding an Excel table here. But try to manage with a simple table – it makes your document readable.

Unless covered by an Integration Test Plan and during integration tests, Feature-level tests need to cover the following:

g) Show that the code does what was specified in the software architectural design

h) The hardware-software interface (unless done by another team. E.g. the HW team; if so, state it in the "Test Scope" section).

i) Verify that non-functional requirements are met

j) Verify correct behavior in case of erroneous inputs

k) Verify the product has sufficient resources to support the functionality

l) Verify the implemented safety measures

The example in the table is for a 3D scanning application.

Table 6: Features to be Tested

Feature / Capability	Additional breakdown	Owning team
Features		
Scan	Face	CVLab
	Torso	CVLab
	Object	CVLab
	Scan configuration controls	CVLab
	Re-acquisition	CVLab
Edit	Crop	CVLab
	Erase	CVLab
	Solidify	CVLab
	Smooth	CVLab
	Trim	CVLab
	Touchup	CVLab
	Edit configuration control	Acme Test house
Save, Upload, Print	Save	Acme Test house

	Upload	Acme Test house
	Print	Acme Test house
	Configuration controls	Acme Test house
Application flows	Main flow	Acme Test house
	Alternative flows	Acme Test house
Documentation	User Manual	Acme Test house
KPIs		
Accuracy	After capture After each edit operation	CVLab
Performance	During Capture During Edit	CVLab
Quality Attributes (non-functional)		
Robustness	Lack of system resources	Acme Test house
	Missing dependencies	Acme Test house
Power states	Standby	Acme Test house
	Hibernate	Acme Test house
	Reboot	Acme Test house
Installation	Install	Acme Test house
	Uninstall	Acme Test house

Do not turn this table into very detailed breakdown of every minute item that needs to be validated. Such list is too detailed for a test plan and is hard to manage in Word. It's helpful to make such a detailed breakdown, but using Excel. If you did such a list in a hierarchical style, often the headers for a hierarchy of details will fit nicely here as Test Modules.

At MTP level, the "Feature" column is for listing the high level features of the product. The test plan for each of these features will be detailed in a separate Feature Test Plan.

At Feature level test plan the "Feature" column is used for listing of Test Modules.

The "owning team" information is of special benefit when testing is split across more than a single team.

Documentation to be tested

This section is relevant for Master or Feature test plan only.

State what documentation is delivered with the product and needs to be validated.

Examples:

- API manual
- User manual
- Get Started manual

In the context of test plan, when we say "documents to be tested" we specifically don't mean "ISO 26262 Work Products". We mean documentation that users get as part of the product and contain instructions how to use or install the product. If these documents are listed in the Software Verification Plan, you can just reference the plan. Otherwise, list the documents here.

Units not to be tested

Identify any code that is in the "test items" folders, but will not be unit-tested. Give an explanation why this is OK. See examples in the table below.

Table 7: Units not to be tested

Code that will not be unit-tested	Reason
Common .h files used by the FW Update tool	Covered by the OS Core team
Service routines called by the FW Update code	Stubbed out by the unit test tool
Files in folder XYZ	Not to be included in this release
Files in folder XYZ	Third party code software that will not be tested by our team
Files in folder XYZ	Unit-tested by other team at your company

Integration items not to be tested

Identify any significant items that are not tested during integration test. List only stuff you are concerned that, unless stated clearly, stakeholders will assume are covered by your team. Explain why these items (interfaces / areas / items / modules) are not covered by you. See examples in the table below.

Table 8: Integration items not to be tested

Modules not to be tested	Reason
Mouhid.sys driver-HW	Already tested thoroughly on previous release; no code change done on this driver for the current project and no change to the HW device.
Module X	Not to be included in this release
Module Y	Low risk, has been used before, unchanged in the current release and is considered stable
Module Z	Tested by other team at the company

Features not to be tested

Identify any significant item/features/configurations that are not tested. List only stuff you are concerned that, if not stated clearly, will be assumed to be covered by your team.

Explain why these features are not covered by you. See examples in the table below.

Table 9: Features not to be tested

Feature / Capability that will not be tested	Reason
User Experience	Covered by the Ux team in the US
User Manual	Covered by SSG
Localization	No localization is planned for the current release
QoS support	Not included in the release for this product
HW/SW interface	Tested by the SV team
Feature X	Not to be included in this release
Feature Y	Low risk, has been used before and is considered stable
Feature Z	Third party software that will not be tested by our team
Feature W	Tested by other team at your company

Assumptions, Dependencies, and Constraints

The borderline between Assumptions, Dependencies and Constraints is not always clear. As an example take this entry:

"Successful execution of this test plan depends on the availability of an Acme Bus Sniffer by pre-alpha time."

Depending on the situation and how you word it:

- It is an assumption: I assume that the Acme company, who committed to have the tool released by pre-alpha, meet their deadline
- It is a dependency: I can't start before I have this tool
- It is a constraint: I must use the Acme sniffer because it's the only one that is capable to measure what we need to measure

A general guideline is to have as a dependency something that is a deliverable of another team (internal to your company or external), while assumptions are for things that are pretty much out of our control (e.g. "WHQL test suite for this new technology is available from Microsoft before WW X").

Constraints are things that narrow the choices you have and forces certain limits. These limits can be on timelines and resources. They may also be limits on choice of test approach, strategy and design.

The important thing is to mention this item as something that progress is linked to. It's not worth spending time arguing if you put it in the right category.

Assumptions

Identify key assumptions and activities beyond your control upon which successful execution of this test plan depends.

Examples:

- Availability of software or tools from an external resource
- Access to 3rd party code
- Availability of <some new hardware> by <some work-week>
- MC/DC coverage feature is available in the next release of the code-coverage tool
- FuSa qualification of the tool used for unit testing is available at the time we start using it
- Support for legacy Microsoft operating systems is not required

Dependencies

List dependencies on other groups. This includes pre-requisites and any other dependencies.

Examples:

- Resources for this project are available after Project X is complete. If major slips to Project X happen, it will impact the resources allocated for this project.
- Availability of simulator or emulator for the HW
- Test tool ABC needs to be updated by team Y to run on the new platform

Constraints

Identify constraints that bound the scope of this validation plan. That is, it narrows the choices you have and forces certain limits on timelines or resources.

Examples:

- Unit tests design must allow execution by the Nightly Build system
- The maximum size of the instrumented code generated for code coverage is 1.5M, to be able to fit into the available FW flash memory
- Unit tests must be created using the LDRA tool suite
- The certification test-house selected for this project accepts submissions only on the first week of every month
- Access to external equipment is available only during weekends
- Due to safety regulations, only 3 testers can be in the lab at any given time

Test Approach

Test Strategy

In this section you discuss the general strategy for testing your code.

Unit test

If the general strategy is similar to that of the [OTS], this section can be rather short. If you deviate from the general strategy, explain here how and why.

Explain the main decisions you made that influence the whole test activity:

Example for stuff to discuss here:

- Do you test in complete isolation or only partially? (E.g. driving inputs, but counting on dependencies to be available and working).
- What is a unit? (Function? Class? File?)
- Who develops the unit-tests?
- When are unit-tests run and by whom?
- What is the tool or framework you are using?
- Do you use code injection or fault injection techniques to activate certain area of the code?
- What are the code coverage goals?

If there is no clear coverage goal, which is probably the case for a non-FuSa code, state this clearly. In this case, the Test Completion Criteria becomes that much more interesting… how would you know when you are done if you have no clear coverage goals?

It is possible that the answers to the above examples differ for different parts of your code. For example, you may select the same approach to all user-mode code (applications; user-mode drivers), a different approach to kernel level and yet another for FW.

Generally speaking, you won't have a per-feature unit level strategy except in special cases. If you do, state so clearly. Similarly, if you have different strategy for safety features VS non-safety features, state so clearly.

In the (unlikely) case where the unit test strategy differs between different features or if you want to clearly separate the strategies for different parts of the code, consider adding sub-sections to the "Test Design Specification" section. See this section in the Integration or Feature test plans templates and decide if adopting similar breakdown works better for you.

Explain the demarcation between unit and integration (or even feature) tests. Is there a conscious overlap? What may be naturally expected to be tested at unit level, but you decided to leave to later test level? Why?

Integration test

If the general strategy is similar to that of the [OTS], this section can be rather short. If you deviate from the general strategy, explain here how and why.

There are cases where this section is all you need to clearly articulate how you do integration tests. This is for cases where the overall strategy is the same for the integration of all the parts (modules; components) that together make your delivery.

If you have a number of different strategies, depending on the integration items, you can talk generally about them here and go into further details in the Test Design Specification section. For example, integration testing may use different tools and different test techniques to test a SW-HW interface and to test an interface to a remote server via REST API. Give a general overview in this section, before diving into further details in the Test Design Specification sections.

Explain the demarcation between integration and feature tests. Is there a conscious overlap? What may be naturally expected to be tested at Integration level, but you decided to leave to Feature test level? Why?

Master | Feature test

Master test

Change the section title from "Master | Feature Test Approach" to "Master Test Approach"

Explain the general approach your team employ to test their part of the project. In many cases, this is already covered in the [OTS] so you can just reference it here.

Additionally, describe your team's role in the overall picture of this project validation effort. Are you the last team to test this project? Maybe you are just contracted to cover specific areas? Describe this in plain language so that other teams (and the project managers) know what you think you are supposed to do for the project and can discuss with you if they think you got it wrong.

Feature test

Change the section title from "Master | Feature Test Approach" to "Feature Test Approach"

Describe the overall test strategy for the feature at hand. If the strategy is not different than the one described in the Master Test Plan (MTP) or in the OTS, you can reference it here instead of repeating everything. Note however, that even when the test strategy for a feature follows the strategy described in the MTP or OTS, you still need to say something about the strategy at the Feature level – even if it is just to explain how it aligns with the higher level strategy.

If the test strategy differs between Test Modules, elaborate more on the test strategy in the Test Design Specification sections. A common situation is that the general test strategy tells part of the story and a more detailed explanation of how this generic strategy applies to each Test Module is covered in the Test Design Specification sections.

Example for stuff to discuss here (applies for both Master and Feature test plans):

- Primary focus of testing (e.g. functional testing; reliability testing; PCs; phones; Android OS; usability; accuracy). There are probably more than just one vector of focus.
- The type of test approach you use and why this applies best here (e.g. use case testing; risk-based testing; requirement testing; regression-averse; model-based; etc.)
- Metrics (general description only; no need to explain the metrics in details here – there is a special section for it)
- Type of test techniques employed to generate the test cases
- Pre-silicon test approach: what is tested and what is not; how is the HW simulated.

No need to cover automation strategy here (it has its own section) – but you should make a comment on manual VS automated testing – how much you plan to automate.

For small projects, it may make sense to generate one test plan that covers both the overall testing picture ("master test plan") and the more detailed, per-feature test strategy. In these cases, use this section for both – first give an overall, project level strategy, then refer to specific features if the test strategy for them differs from the project-wide strategy.

Safety requirements test strategy

If there are no safety requirements in this feature, put "NA" here and remove all subsections to avoid clutter.

For Safety related code or features, there is plenty that you need to provide in this section. The ISO 26262-6 standard contains several tables that enumerate testing methods and the ASIL level that requires each. The template contains references to these tables.

Note that you may have situations where it is proper to add the ISO 26262 tables again (or only at) the Test Module level. This is for cases where one type of test technique is applied to certain aspect of the tested code, while for a different aspect, a different test technique is used. If you go this way, make a note in this section that "further details that apply to each test aspect of the code are detailed in section X".

Test Completion Criteria

Also known as "Exit criteria".

How would you decide that you are done testing? The general answer is: When the needed coverage is achieved and all tests are passing. So if you don't specify it elsewhere, this is a good place to declare the coverage goals.

Unit Test

Examples for Pass and Fail criteria for unit testing (Select those that apply and/or write others similar to these):

- 100% of the unit test cases pass
- All unit test cases dealing with critical functionality pass
- All medium and high severity defects are fixed
- Sentence coverage is 100%
- Sentence coverage is above 90% and all discrepancies explained
- Branch coverage is 100%
- Modified Condition/Decision Coverage is 100%

Integration test

Example for Pass and Fail criteria for integration testing (select those that apply and/or write others similar to these):

- 100% of the integration test cases passed
- All integration test cases dealing with critical functionality passed
- All high and critical severity defects are fixed and verified
- Function coverage is 100%
- API coverage is above 90% and all discrepancies explained
- Call coverage is 100%

- Requirements test coverage is 100%

Feature test

Example for Pass and Fail criteria for feature testing (select those that apply and/or write others similar to these):

- 100% of the test cases were executed and 97% of them passed
- All test cases dealing with critical functionality passed
- All critical and high severity defects are fixed
- Requirements test coverage is 100%
- A certain number of test cycles with incoming bug count trending down
- A certain number of test cycles with no new critical bugs

If your company has a standard set of Release Criteria, you can just reference that, at this test level.

Suspension Criteria and Resumption Requirements

In case you have such criteria, list it here. In my experience, the situation is usually a case-by-case decision, made by validation managers in consultation with the validation feature owners and the program manager.

Note that for ISO 26262 compliance you may not be able to get off so easily and will need to list something more concrete. Consult the expert in your company.

Examples for Suspense criteria:

- A Change Request is approved that requires significant architecture, design or code changes.
- The amount of bugs in a specific part of the code is above a certain threshold, rendering further testing useless since the code will anyway need to go through significant changes
- The system is too unstable (e.g. resets itself every 2 minutes) for proper use of the testers' time.
- The released version fails the Build Acceptance Tests

Resumption requirements are usually the activities needed to make the suspension criteria pass.

Configurations coverage strategy

Start with just listing the platforms that are used in testing, the OSs and (when applicable) the SKUs. Continue with a list of the combinations that are covered. This can be either an exhaustive list, a verbal explanation, or a URL to another file containing the full list.

Then explain the strategy and the considerations used when specifying the covered combination list.

The coverage may be different for each test level.

- For unit tests, especially when tested in full isolation, this section may be pretty much redundant as the tests are executed on whatever platform that is available to the developer of the CI automation. If the code is different for different OSs, this section becomes more relevant.
- For Integration test, it is common that testing is done only on one platform – especially when testing SW-only interfaces. If you are testing interfaces with the OS, you may want to cover the target OS list. If you are testing integration with HW, you may want to cover all the relevant HW flavors.
- At Master Test Plan level, list all the combinations covered and specify the priorities among them. This gives the Feature Test Plan writers a direction when doing their own prioritization.
- At feature level, the decision if to cover all or just some of the combinations mentioned in the MTP is feature-dependent. When covering all the combinations, it is common to select some combinations as the primary ones and these get more testing. If the coverage strategy detailed in the Master Test Plan is adequate for a feature at hand, it's enough to reference the MTP.

Platforms and OSs coverage

Platforms

List the platforms you test on. List also SKU if relevant.

OS

List the Operating systems you test on (with version number where applicable).

Example:

- Windows 10 (from build number 10538)
- Android 11.0.0

Platform and OS combinations coverage strategy

If you have more than a single Platform-OS combination, you need to decide how testing will be spread over the combinations. The brute-force option is to test everything on all combinations and in some cases this is the right approach. But this could mean a LOT of testing – especially if you have many

combinations. For situations where covering all combinations becomes more testing than you can possibly execute, you have to define some strategy – what gets covered on which system configuration.

If your product has SKUs, you need to describe how you deal with that as well. Is there a primary SKU? Is the feature discussed in this test plan not impacted by SKU, therefore it does not matter? Etc.

A common approach is to select a "primary" configuration and this is the one that sees most of the testing, while other configurations are being tested on features you know (or suspect) behave differently on these combinations. If there is no such expectation (that is, you think the behavior will be the same on all configurations) then other configurations will be getting a low level of sanity tests just to make sure you don't have a glaring, configuration-specific bug.

In other cases you may be able to cover all combinations by covering different combinations each test cycle.

Yet another approach is to test some of your test modules on many configurations, and some only on one or two.

It is also possible, for the case of Trunk Based development, to rely (for coverage) on tests done on other projects altogether, provided they use the exact same code as your project does and when the features are HW independent. This will reduce the level of testing done on this project. This of course calls for tight coordination with the other projects teams to ensure you don't open gaps.

The exact test effort distribution over configurations is dependent on your feature set and the program goals – and this is what you need to discuss in this section:

- What combinations you will cover
- How much testing on each combination
- The rationale behind these decisions

☑ Sometimes the same platform has a number of configurations that need to be tested (e.g. RAM, Graphics, Camera version and any other HW or SW requirements for the platforms). If this is the case, list these configurations, and what combinations of platforms - configurations - OS you cover in your tests.

☑ If there are many Platform-OS combinations with much detail, consider adding a table here, or even embedding an Excel sheet.

☑ For Feature Test Plan: Some projects may have the configuration discussion covered in the Master Test Plan; in these cases you can reference that document and just note here if some configurations are not relevant for this specific Feature Test Plan.

[C&C] is a good reference in case you have no experience in defining a test strategy for many Platform-OS combinations.

Software configuration and calibration coverage strategy

This section is mandatory for code that includes safety requirements. It is worthwhile thinking about the topic even if your code is not safety relevant. [C&C] provides a suggestion for test strategy of the software configurations and calibration parameters.

One common approach for creating different flavors of a product, using a common code base, is what ISO 26262 refers to as "Configuration". These are, for examples, compiler settings which would include or exclude certain parts of the code in the compilation. The resulting binaries are obviously different.

On top of configuration – which impact the actual binary of the product – ISO 26262 refers to "Calibration data". These are the settings applied to a binary and control its behavior. A simple example is the enabling or disabling of "line number" feature in a text editor.

Per ISO 26262, there are four levels of verification:

- Verification of configuration data
- Verification of configurable software
- Verification of configured software (the result of applying configuration data to a configurable software)
- Verification and validation of software (the result of applying calibration data to a configured software)

Before discussing the coverage strategy for Configuration and Calibration, first list what configuration and calibration data are relevant to the code under test and what combinations are covered. List all values or reference the "Configuration data specification" and "Calibration data specification" ISO 26262 Work Products. After listing the data and the combinations that are covered, explain why these combinations are covered and why the risk of not testing other combinations is acceptable.

Explain how the four levels of verification are covered. It is possible that this is already covered in the OTS, so reference that document if relevant.

☑ To get the full story on Configuration and Calibration, see ISO 26262-6:2018 Annex C.

☑ See [C&C] for a proposed test strategy for testing Configurations and Calibrations

Test Tools & Automation Strategy

What is the overarching automation strategy for unit test? For Integration? For the feature that this test plan cover? For the whole project (for MTP)?

Enumerate the test tools you intend to use and explain shortly what they do. This may include items such as internal frameworks, specific tools written to test your feature, specific 3rd party test harnesses or tools etc. In many cases this will be the same as in the OTS, so just reference that document.

Will you need to invest money or resources?

Clearly state which of the tools are existing ones and which need to be developed for the implementation of this test plan.

Unless the OTS covers the use of project-level tools such as bug tracker, requirements management tool, etc., they will need to be mentioned here.

For ISO 26262 compliance, ensure that this section covers the following:

- What tools will be used for verification (if applicable)? [ISO 26262-8:2018; 9.4.1.1f]
- For FuSa, the SW tools used in the validation process may need to be qualified. See ISO 26262-6:2018, Section 11 and 12.4. This would be covered in the Tool Qualification work products – but keep this in mind and ensure these tools are listed in the tool classification list and that you know if they need to be qualified.

Specific information relevant to Unit Test Plan

What overarching automation strategy are you using the Unit Test? The simplest thing is that every developer runs their own tests on their development machine. But you can also opt to run unit tests as a pre-check-in gate on in CI.

If you collect code coverage information, are you collecting it for each test or for all tests together? Collection for each test probably calls for some automation of saving intermediate files and for running the analysis. You will also need to have a setup where all the tests are running together, so that the overall coverage can be measured. Alternatively you will need to set up a reporting mechanism where each developer reports coverage and evidence, so that the project-wide coverage numbers can be generated.

Test Design Specification

This section provides the information that ISO 29119 specifies for the Test Design Specification document. It contains also some of the information that is in the 29119 Test Case Specification document.

The Test Design Specification section contains a varying number of Test Module sub-sections, each dealing with a specific aspect of the product. The partitioning of the code-under-test into different aspects vary between unit, integration and feature test plans. See further explanation on each test level, below.

At Master Test Plan level, this section is in most cases “NA”, as test design is usually different for different features.

Unit test

For unit tests, the "Test Design Specification" section may have only one Test Module, as usually tests follow the same approach for all the code. If this is not the case, consider adding additional Test Module sub-sections as needed.

Unit tests are intended to verify that the software units:

- Comply with the software unit design specification
- Comply with the specification of the hardware-software interface
- Deliver the specified functionality
- Do not suffer from unintended functionality
- Are robust (deal correctly with all error cases or invalid inputs)
- Have sufficient resources to support their functionality.

The description should be general. There is no need to give details on each test, only about the type of tests you employ. In many cases this will be very generic and applicable to any project.

Integration

Depending on how you define "integration tests", the content of Test Modules would vary.

In the context of interface testing, a Test Module would detail how tests are designed for a specific type of interface. The same test design approach would be used for this type of interface – wherever it appears in the product.

If your org uses Integration Tests for something other than interface testing, make your own definition of what each test module covers. In some cases the definition defined for Master | Feature Test Plan may be right for you – in which case you should seriously consider using the Master | Feature Test Plan as your template. If all integration tests are designed in the same way, you may have just one Test Module entry in the Test Design Specification section.

Feature

At Feature level, write a Test Module section for each line item in the "Features to be tested" table.

<Test Module #1>

The name of the test module.

Unit and Integration

The name should give the reader an idea what type of unit or integration testing is covered in this module.

Example (for integration tests focusing on interfaces):

- API

- Windows IOCTLs
- REST interface

Feature

The name of the test module. This should be one of the entries in the "Features to be tested" table.

Description

A short description of the test module: What aspect of integration tests or the feature is covered in this Test Module. Two-three sentences are usually enough.

Test Strategy & Validation Method

Describe the method used to validate the aspects assigned to this test module.

When done reading this, the reader will know how you selected the specific tests you have from the infinite number of existing tests – and why you decided to go this way; why you think this selection gives good coverage at an acceptable cost and effort.

For ISO 26262 compliance, ensure that this section covers the following:

- What techniques/methods will be used for verification? [ISO 26262-8:2018; 9.4.1.1c]
- Why are the verification methods planned adequate for the verification activity? [ISO 26262-8:2018; 9.4.1.2a]
- What are the specific methods/strategies/activities that will be used for verification of the correctness and consistency of the work product with respect to its input? Why were they chosen? [ISO 26262-8:2018; 9.4.2.1]

Unit test

The text in black italics below is a good starting point for strategy and validation methods for different unit test types. You can opt to list all test types together (only one Test Module under the "Test Design Specification", with possibly some sub-Test-Module sections) or you may use the structure of Test Modules to discuss each test type separately and in more details.

Input validation

Each function is tested for correct behavior with both valid and invalid input values. When relevant, these values are selected to be at the boundary values of the input parameters equivalence classes. In this context, global variables in use by the unit-under-test are also considered "input".

When using boundary values to select test inputs, each input parameter is tested at the boundaries while the other input parameters (if exist) are at nominal values.

Error codes

Specific error cases are generated, to ensure that the correct error code is returned in response. This covers both errors resulting from invalid input (the result of input validation by the code-under-test) and errors returned from functions called by the code-under-test. Stubs or mocks of the called functions are used to simulate these error cases.

System calls

For positive tests, system calls are generally used as is (not mocked), so that access to the file system and system resources other than the HW under test are available. Some special cases may require replacement of a system call with a mock.

For error codes tests, system calls are mocked to create the desired error situation.

Functionality

Tests are validating that the functionality of the unit under test is as defined.

Integration and Feature tests

Describe the methods and test techniques used to validate the aspects covered by this test module. If there is no specific thing to say about this module (that is, you use the same approach already explained in the Integration | Feature Test Strategy) you can be very brief here – as brief as "Follow the same approach as outlined in the Integration | Feature Test Strategy section".

Mention any specific considerations and how they are addressed.

If you have special debug or testability functionality, explain how you verify that this functionality works before using it in testing and how you ensure (test!) this functionality is removed or disabled in the final product.

If you use specific test techniques in the test design for this test module, specify which and how the techniques were used to create the test list. Explain why this is the right test technique to use.

Explain what validation methods you apply (e.g. how the tests are executed; how the results are analyzed to decide the test outcome; do you use automated comparisons; manual verification; post-processing of any sort; etc.)

When you cover a combination of inputs and environment variables and need to deal with "combinatorial explosion", explain the method you used to achieve adequate coverage (e.g. all-pairs; base choice, etc.).

For flow testing: why did you decide to cover certain flows and not others? This is especially important for tests that are calling APIs using a certain order; if a calling application may achieve the same functionality using different API call order you need to decide what API call flows are covered and what are not.

Sub-Test-Module A

Sometimes a test module includes many items and it makes sense to have further sub-sections here. If needed, add Sub-Test-Modules. If you have "Additional breakdown" in the "Features to be tested" table and each of the items in the "additional breakdown" is a rather complicated item by itself, it's a good indication you need Sub-Test-Modules.

- In most cases there is no need for sub-feature breakdown. In this case remove this sub-section and continue with "Test Steps"

Sub-Test-Module Test Strategy

Add details if the strategy is different than the one listed for the whole test module. Otherwise just write "No change from the Test Module strategy".

Test Steps

In most cases, when the partition into Test Modules was selected correctly, the module's tests will all look more or less the same, with the difference being in the actual input values used and the expected results. So the "test steps" are really a generic description; all the tests for the Test Module will follow the same steps.

If you find that the tests don't follow the same generic steps – you may have lumped more than one aspect of the feature into this Test Module. Maybe it makes sense to split it into more modules? In some cases further splitting makes sense, in others not; do your own thinking and decide. You can opt to keep the module intact and have more than one generic set of test steps. No big deal.

Be extra short here. Describe the tests in very few words (e.g. "Apply invalid values to each API input; verify correct error code") – no need to write actual steps in most cases. When it feels natural to write the test description as more detailed steps – go ahead. But do keep it short!

Use a table if it helps.

Specify the test environment to be used (use the names you defined in the "Test Setup" section). There is no need to list trivial steps such as "install this" "install that".

You need enough info so that readers will understand what the tests do and how they are constructed. Since no one will use this document as step-by-step instructions how to run a test, save the excruciating

details to the place where you specify test cases and test steps (e.g. a test management system; or an Excel file; or whatever).

If the tests are run by an automated system, indicate so. Consider if to give the command line that will run the tests. If all the tests in the module are executed by this one command, it may make sense. Otherwise, give the details as part of the detailed test cases.

Example (manual test)

Test environment: Basic setup (for an IoT device with peripherals)

- Build an image following to the specifics in the test case
- Burn the image and boot
- Check for correct driver and sub-system status (success for positive tests; fail for negative tests)
- Repeat for:
  - Camera
  - Audio
  - WiFi
  - Type-C connector

Example (automated test)

Test setup: BAT setup

- Set the WiFi card status to one of the airplane modes (On or Off)
- Execute power cycle
- Check that the card stayed in "airplane mode"
- Repeat 100 times, for each power cycle type (S3, S4, S5)

To run, execute:
C:\ powerFlow.exe –c S3 –n 100 –AirplaneMode on

Test Tools

List the test tools that will be used in this test. No need to explain anything on the test tools – you already did this in the "Test Tools & Automation" section Just write their names.

Example:

powerFlow.exe

Restrictions, Limitation and Exclusions

Describe any restriction, limitations or exclusions that should be known about this test.

Examples:

- The test runs only in specific configurations
- Access to the certificate authority (CA) server is restricted from the lab environment. The test, therefore, simulates the CA server. We run the test manually, for Beta, using a direct internet access link.
- The system must be rebooted at the end of each test

Test Cases

Give a pointer to where the test cases are.

- For unit tests this will usually be a path on the code repository system.
- For automated tests that are also tracked via a test management system, give both: a URL to the test management system and the code location in code repository.
- For Feature this will usually be a path to test cases in a test management system.

For automated tests, you need to explain how to compile the automated test code, how to run it on the target code and where the results (including, when applicable, performance and code coverage results) are to be found. The correct approach is to have a reference document where this is explained.

Explain where each test case is described. For automated tests, the recommendation is that each test is described by a comment at the head of the test code. If the test is defined in a test management system, the test should be described in that system. If tests are using different setups, test description must include a reference to the setup used.

Tests should be traced to requirements. This is a must for safety-related features and highly recommended for any other feature. Explain how tracing is done and where it can be viewed.

For ISO 26262 compliance, ensure that this section covers the following (in most cases, coverage will be by the details in the test case management system):

- What are the pass and fail criteria and process for evaluation of verification results? [ISO 26262-8:2018; 9.4.1.1c]
- Complete a), b), c) or a combination of a), b), and c). [ISO 26262-8:2018; 9.4.2.1a, b, c]
  a) Provide review or analysis checklist to be used in verification and rationale for sufficiency
  b) Provide details of simulation scenarios which will be used in verification and rationale for sufficiency
  c) Provide details of the test cases, test data, and test objects to be used for verification and rationale for sufficiency

If testing is applied (Option c) above), provide the following details for each test method applied (group of test cases) [ISO 26262-8:2018; 9.4.2.3]

a) Test environment to be applied per test method
b) Logical and temporal dependencies per test method
c) Resources required for each test method (tools, setup, etc.)

- Coverage of requirements at the software architectural level by test cases shall be determined. [ISO 26262-6:2018; 10.4.4]

Testability Hooks

List here the testability features you need in order to be able to execute the test cases you plan. Write only those features that already exist or that development agreed to develop; this is not a wish list. If you find, through looking at the ideas below, that a test hook will make your life simpler, make the test shorter or allow testing something you can't test today… go talk to the developers and try to convince them to implement it. Once they do, you can list the hook here.

If you test your feature without any test hooks – just write NA in this section.

Note: In some cases, testability hooks are added to intermediate, development versions, but removed from the final product. This means that the tested SW is not exactly the same as the one eventually released and that some tests cannot be run on the final binary going out. You should consider the implications. Does this add unacceptable risk to the released code? There are several possible approaches:

- Leave the testability hook in the final code (adding, sometimes, a security risk or exposing information you don't want to expose)
- Analyze the risk and decide if it is acceptable
- Find a less intrusive way to test the code (maybe in a less efficient way – which may be acceptable if it is only for the last test cycle).
- Etc.

ISO 26262-6:2018, section 10.4.7, requires documented analysis of such situation. See the full text in "Test Design Specification" section.

The following is a list of testability features to think of. Some may be relevant to your feature. Asking for these on time (during requirements and coding phases) increase the chance of getting them. You can think of other hooks too!

Source: "Design for Testability" – lecture by Bryan Bakker, QA & Test 2010 conference

Think of:

- Testing without HW

- Simulated environment (for automatic testing or unfeasible environment)
- Testing without physical access to the DUT (workaround mechanical switches/buttons to allow non-attendant testing, to enable test automation; e.g. a way to power / shut down the system remotely)
- Support for test automation
- Negative testing (failures from HW or environment)
- Support for test/SW engineers (diagnosis)
- Logging/Tracing
- Test components in isolation (modular architecture)
- Support for integration testing (test for messages)
- Test without UI
- Reliability/Profile testing: record user actions and replay

Achieve the above by:
- Visibility
- Ability to control

Ability to extract all kinds of system information. E.g.:
- Temperature
- Recording speed of recorder
- Mechanical movements verification
- Inspect messages (for integration tests)
- Logging (better inspection/analysis, tool support)
- Resource usage (CPU, memory, network)

Ability to trigger all kinds of system actions
- Push buttons (UI, mechanical)
- Set configurations
- Simulate events (motion events, alarms, hot temperature)
- Mechanical movements
- Simulate HW failures/imperfections
- Flash ("burn") a new FW

State visibility:
- Ability to get state information from each component
- Ability to dump the complete system information
- State machines trace/log state transitions

Communication between each set of components:
- Visible (minimal requirement)

- Controllable (less critical – depending on the case)
- Via dedicated or standard interfaces

Ability to log user actions

Ability to record and replay the user's actions.

Test Environment

This section covers both the details of a test bench setup and general lab environment needs.

For ISO 26262 compliance, ensure that this section covers the following:

- Describe the verification environment. [ISO 26262-8:2018; 9.4.1.1d]
- Discuss the test environment relative to the requirements of [ISO 26262-6:2018; 10.4.7]. If you are testing the code loaded to the target processor without any simulations involved, you are not running any of the options listed in 10.4.7[Note3] - and this is OK since we actually prefer testing on the real final HW. Otherwise, apply SIL, MIL, PIL, HIL to have a test environment as much as possible similar to the real final environment. If not possible, highlight the differences so that the subsequent test phases can be fine-tuned accordingly.

Test Setups

Describes the test environments or configurations to be used during the execution of tests.

For unit tests in complete isolation, this is probably just the standard development machine setup. For unit tests that are performed on the target HW, this will be more complicated.

At Master Test Plan level this section may be NA (when the setup is different for different features; there is no point in listing all the setups at MTP level). On the other hand, if the same setup is used for all the project, it can be described in the MTP and save the need to repeat it in all Feature test plans.

Give each environment or configuration a unique ID (e.g. CHL_1). The test cases can then reference a specific setup ID instead of specifying the setup details.

If you are testing on more than one OS – you need to specify a test setup per OS.

Ideally you should use setups that match those that the final released product will be used on. But there are cases where proper testing cannot be done on the production version or environment and you need to use setups that differ from the final released product.

Examples:

- Pre-production systems that allow access to information that is locked-out on production systems
- Debug setup that allow testing features with limited controllability or observability in the Release version

For FuSa, these cases call for special attention:

For unit tests, ISO 26262-6:2018, section 9.4.5 requires to give explanation if not testing on the target environment. If you test in isolation, you will not be using the target HW since no HW is needed. This is a good enough explanation… But if you are using a simulator instead of the hardware, or a hardware that is not the final target hardware – you need to state so and justify why.

A similar requirement exists for Integration and Feature tests, in ISO 26262-6:2018, section 10.4.7: If the software integration testing is not carried out in the target environment, the differences in the source and object code and the differences between the test environment and the target environment shall be analyzed in order to specify additional tests in the target environment during the subsequent test phases.

If this test plan document contains more than one level (e.g. Unit test & Feature test) and the setups are similar, it's enough to describe the setup once, and reference it (by setup ID) in the other section of the test plan.

When defining test environments for a test, always specify the simplest setup that allows running the specific test. The tester can always decide to build the most complex environment to allow running all tests on the same environment. However, if you need to split tests across several setups, you will probably not want all the setups to be the complex one, for cost and setup time considerations. Specifying the simple setup will allow building what is needed and not more.

Setup 1 [Setup ID]

Duplicate this section and its sub-sections as many times as there are test environments.

Environment Diagram

In most cases, adding a setup diagram makes it much clearer. Before spending your time drawing setups, check if it is possible to cut-and-paste the setup description – or at least good parts of it – from other test plans or from a PowerPoint presentation.

<Add a diagram here >

Environments Components

The components appearing in the above diagram are described in the following table:

Table 10: [Unit | Integration | Feature] Test environment components

Add rows to the tables as needed

HW Components	
Equipment Name	**Details**
Base Platform	OS, memory config if relevant, platform camera, etc.

SW Components		
Type	**Name**	**Version**
Screen Recorder	Camtasia	The version of the SW component installed on the DUT or on other environment ingredients

Setup assembly instructions

Add here instructions for assembling the test setup. This can be anything from a very simple "Install all SW components" to details about HW connections and wiring, order of SW components installation; per OS instructions; workarounds; troubleshooting etc. The level of detail needs to be such that a person with relevant background in the tested product and technology will be able to do the setup by themselves. It often happens that at the start of the project the instructions are complicated, calling for manual steps and temporary solutions. Later, the setup simplifies. Write what is correct for now and update the instructions as they change during the project.

You can have a URL here, pointing to an online resource (e.g. WiKi) where the setup instructions exist. But you then need to make sure no one breaks the link. So maybe just copy/paste the instructions here.

Hardware and Lab

List here items that are needed to execute the test plan. How much lab space and lab tables do you need? Do you need racks of equipment that take space and take time to set up? A large cluster of machines that calls for a high capacity air-condition unit?

Pay special attention to long-lead-time items and to expensive items – these may take time to get.

For unit test, this section will almost always be "NA".

Software Environment / Environment configuration

Do you need special operating system flavors? How would you get the latest OS updates? Are there software tools that are not part of the immediate setup but are needed for the tests? Do you need to buy licenses? (Examples: test execution tools, network servers, authentication authorities, databases, utilities of all sorts, statistics analysis packages, unit test framework, etc.)

In many cases this will simply be:

Nothing special over the details in the Test Setup section.

Examples:

- For testing Android you may need an external control system
- Do you need a certificate from some certificate authority?
- DHCP, DNS and others such servers

For FuSa, the SW tools used in the validation process may need to be qualified. See ISO 26262-8:2018, Section 11.

Security & Privacy

If you use test data that contains personal information (images of people; other personal data), explain how you follow the required legal processes.

This is NOT the place to talk about security testing (as in cyber security). If your feature or product has such considerations and you are testing for that, this should be an added Test Module in the Test Design Specification section (and in the "Features to be tested"). Possibly it will merit a Security Test Plan – which is not covered by this document.

Test data requirements

Are there any special requirements for data? In some projects, this is very critical and there is a lot of work associated with generating the test data (e.g. when this data needs to be collected in the real world; when a large database needs to be populated for test purposes; when ground-truth has to be prepared manually). In other cases, there is hardly anything to say here ("NA" is a good choice in these cases).

Test Execution

Test Entry Criteria

When will you start testing? What's the minimum that must be available or the minimum functionality that the code must show before you invest your precious time to test it?

BAT Strategy

BAT (Build Acceptance Tests) tests are also known as "Smoke Tests"

Do you have a BAT Test? When is it run? By whom? What is the criteria for selecting a test for the BAT suite? What happens if BAT fails? Are there cases where BAT failures do not block start of testing?

☑ This section appears only in the Master | Feature test plan. Depending on what you call Integration tests, you may want to add it to Integration test plan.

Continuous Integration Test Strategy

Unit tests

Is unit testing going to be part of the CI tests? If so, you need to think about how tests are extracted, compiled and applied to the build; where results are collected; what happens on fail, etc.

Integration tests / Feature tests

Do you have CI? When is it run? By whom? What system? What are the criteria for selecting a test for the CI test suites? How often is it run? What triggers a run?

Do you have a gated check-in methodology? What are the criteria for selecting a test for the gated check-in test suites? Are these tests installed on the developer's system or is there a centralized framework? Is it triggered by check-in or is there a need to manually trigger it?

Regression Strategy

What's your regression strategy? Risk based? Kitchen sink?

What are the criteria for adding tests to the regression suite?

Who runs it? How often?

Do you approve taking intermediate SW drops? If you do, what's the testing strategy once an intermediate drop is delivered (do you require to reset and re-test all tests? Some? none?). How often do you expect the test teams to run a test cycle?

How do you prune the regression test suite (how often is it reviewed? What are the criteria to remove tests from the regression suite?)

For ISO 26262 compliance, ensure that this section covers the following:

- What is the regression strategy for re-execution of verification after a change to the work products under verification? [ISO 26262-8:2018; 9.4.1.1g]

Compliance and Certification

Does any feature in the tested code subject to certification, compliance requirement or regulatory requirements? How do you plan to test this? To achieve the needed certification?

☑ This section appears only in the Master | Feature test plan. Depending on what you call Integration tests, you may want to add it to Integration test plan.

Milestone Release Testing

What's your milestone testing strategy? Do you just run all the regression tests or more?

What are the criteria for adding tests to the milestone test suite?

Are there tests you only run on a Milestone release and not in the regular regression test? Why?

☑ This section appears only in the Master | Feature test plan. Depending on what you call Integration tests, you may want to add it to Integration test plan.

Metrics to be collected

What metrics will be collected and tracked?

At unit test level, this is usually coverage metrics. How will you combine coverage data from various sources (e.g. developers) into one report?

At Master Test Plan, these would be things that are tracked on the dashboard for the project (e.g. open bugs; closed bugs; completed tests; etc.).

For Feature Test Plan these will usually be related to number of tests planned, executed, pass, fail. Sometimes you will also track performance metrics that are specific to the tested feature set.

It is likely you already listed the information about metrics in the test approach section. In these cases, just make a reference to the relevant paragraph.

For FuSa, you must provide the measure of requirements coverage by tests. For this, you need to provide a traceability matrix of requirements and the tests that cover them. It is assumed that this matrix is managed in a database or an external table. Reference this here. Note that the material you provide for the "Test Cases" section may be prepared in a way that provides the requirements coverage indicator.

Test Monitoring and Control

How do you track your progress? What triggers update activities (such as adding more tests; pruning test lists; changing test suite content)? Who makes sure that tests are created and executed?

In many cases there is nothing special done – you just follow the project-level or org-level processes. In these cases, just reference the relevant document (e.g. [MTP] or [OTS]).

Bug Management

How do you manage bugs? In what system?

Who runs the SysDebug (or Bug Scrub, or whatever name your org call the Bug Triage meeting)? What is the process used by that team?

How do you deal with unit test and integration test bugs? Do you report them? Where? In many cases, unit and /or integration bugs are solved on the spot – do you report these? Log them somehow?

How will you deal with bugs to 3rd party, and bugs from 3rd party?

If you have multiple branches, do you clone bugs from one branch to the other? How do you ensure that all branches are fixed?

In most cases you can simply reference your organization's bug management processes defined in the [OTS]:

Bug management follows the standard process as defined in the [OTS].

For ISO 26262 compliance, ensure that this section covers the following:

- What actions will be taken if anomalies are detected? [ISO 26262-8:2018; 9.4.1.1h]. The information about bug management may be covered, for FuSa, in the Change Management work product. In that case, reference that document here.

Bug Fix Verification ("re-test")

Anything special? Or just use the below.

Unit test

All unit tests are expected to pass prior to checking in the code. Therefore, re-test is being done on the fly as part of the development process.

Integration test / Feature test

Bug fix verification is the first task done on a new release, before the start of any test cycle.

Test reporting

How are test results and test evidence collected and reported?

If the tests are in a test case management system than reporting is usually a report from the system.

In most cases, this will not be different than what's listed in the [OTS], so this section becomes a reference ("see OTS section X").

You may have already answered this in "Test Monitoring and Control". If so, just reference that section.

Risk analysis

For Safety-related projects, Risk Analysis should be done centrally. If this is how your organization manages Test risks, reference the relevant document instead of filling the table here. Centralized risk analysis is also an option for regular projects.

Identify any major risks to the successful execution of this test plan that are known at the time of writing this plan. For each risk, identify level, owner, contingency plans. State when and how risks will be reviewed. Discuss how the risks will be managed. For example: "Risks will be raised and tracked at the weekly meetings".

For ISO 26262 compliance, ensure that this section explains how test planning was influenced by and how test covers the following

- How was the complexity of the items under test taken in consideration when planning the verification and test activities? [ISO 26262-8:2018; 9.4.1.2b]
- How were prior experiences related to the verification of the items under test taken in consideration when planning the verification and test activities? [ISO 26262-8:2018; 9.4.1.2c]
- How were the degree of maturity of the technologies used or the risks associated with the use of these technologies taken in consideration when planning the verification and test activities? [ISO

Risk analysis.

26262-8:2018; 9.4.1.2d]

Product risks

For Master Test Plan, discuss which features are deemed more important, critical or risky than others (in term of the confidence it will fulfil the functional and non-functional requirements associated with it). Other criteria that can identify risky features:

- New features (VS legacy features)
- Features whose development process used new methods, concepts, tools, or technology
- Complex features
- Features that have high user visibility
- Features with a history of high bug count or high bug severity
- Features who are frequently changed (lots of Change Requests)
- Features that were developed by new / novice developers

For Feature Test Plan: In many cases, some of the Test Modules are more important, critical or risky than others (in term of the confidence the code will fulfil the functional and non-functional requirements associated with them). This should mean it gets more testing. Describe the considerations that lead to deciding which aspect of the feature gets more effort and attention. Discuss also what type of testing (e.g. functional; use case) gets more focus and if you have different coverage goals for different test modules.

For each identified risk, provide recommendations to mitigate it.

Project risks

Identifies test-related project risks and provide recommendations to mitigate it.

Some of the project-level items that can make the test plan ineffective are scope creep, late drops, environment creep, build quality, lack of resources. Discuss contingency plans if one of the risks materializes.

Example for a risk table is below. Note how colors are used to mark high-medium-low risks. Both project and product risks should be tracked in the table. If you feel it will help you, add a column identifying the type of risk (product, project). Personally, I thought this just adds overhead so opted to leave the sample table without it.

If this test plan covers more than a single test level (e.g. you cover unit, integration and feature test plans in the same file), consider adding a column identifying the test level relevant to the identified risk.

If risk is managed formally somewhere else (e.g. in an ALM system) - just reference that system here.

Table 11: Project and Product Risk and Mitigation

Risk	Risk Type	Level	Owner	Contingency	Status
Platform resets after long use	Product	Low	Jane	Hold off running stress tests; rest the platform manually before long tests	Open
No resources to create Ground truth for Face recognition	Project	Medium	Jim	Subjective evaluation of the resulting model Implement some of the ideas listed in this test plan	Open
PDX module is late or only partially functional	Project	High	John	Test the base logic without PDX. All tests will need to be repeated once PDX is functional	Open
Image processing code is too slow	Product	Show stopper	Jo-Ann	Stop testing at 90 fps until the algorithm is optimized	Open

Schedule, Project management and Staffing

General comment: This section was added here since ISO 29119-3 (Software and systems engineering — Software testing — Part 3: Test documentation) has it as part of Test Plan document. ASPICE does not require in its work products list (Automotive SPICE Version 3, Annex B, item 8-52, where it references the ISO 29119). Which means that if your organization does not coordinate and manage staffing and schedule via test plans, you can just reference where this is being done and move on. You can even opt to just write here NA.

HAVING SAID THAT and before you rush to ignore this section: why don't you spend the better part of 5 seconds and think if you have the needed staffing? Do you need to employ 3rd party testers? Are there training needs you need to plan for? Find the answers and take care of the needs. Then, if you manage schedule and staffing somewhere else, just say so here and move on.

Schedule

Add here any schedule information relevant to this test plan.

In most cases, all you will have here is a reference to the place where your organization has schedule information. It is recommend to do so since whatever schedule you put here will not be correct two weeks into the project.

If you insist on having some type of schedule here, make it a high level one, with a clear disclaimer and a link to where the most updated schedule is located.

Link to Project Plan file: Path, URL or other valid pointer.

Disclaimer: *This high-level schedule is probably good for the few weeks around the creation date of this document. Take it with a large grain of salt and ask the project managers for confirmation of dates if you plan anything that depends on project milestones. It is provided here to create general timeline context.*

Table 12: Milestones schedule

Milestone	Date
Pre-Alpha	WWxx
HW samples	WWxx
Alpha to Val.	WWxx
Alpha to OEM	WWxx
Beta to Val.	WWxx
Beta to OEM	WWxx
PV to Val.	WWxx
Release	WWxx

Project management

The [OTS] may already have the information requested below. If so, reference that document.

This section is a good candidate for marking as "NA" unless this is your Master Test Plan.

Project ownership

Who is running the project? How are decisions taken? Who owes information to whom? Who represents the testing teams in the product management team?

Coordination between test teams

What teams are involved? What type of coordination is expected to exist (and be maintained) between the test-teams?

Test Cycle creation and tracking

How will test cycles be created? Who decides their content? Based on what? Is there a single cycle for all teams, or a cycle for each test teams? If a single one, how do testers know what their team owns? Do you copy results from previous cycles? When are all results being reset?

Deciding test-cycle frequency, assigning resources and tracking execution is normally done via other systems. Put here a reference or a URL to the system used by your organization.

Standard meetings

Are there any standard meetings for this test-project (e.g. weekly validation meeting; daily bug scrub)?

Staffing

Generally, staffing plans are not done in the test plan documents but by the project managers, using other documents and systems. Note though that SOMEWHERE you should have a coherent list of who does what – and this list can be referenced here.

You may want to write here something generic about how and where staffing is dealt with in your organization.

Roles, activities, and responsibilities

Quoting ISO 29119:

"Provides an overview of the primary (they are the activity leader) and secondary (they are not the leader but providing support) people filling the test-related roles and their corresponding responsibilities and authority for the testing activities. In addition, identifies those responsible for providing the test item(s). They may be participating either full- or part-time. "

Example:

The responsible parties could include the project manager, the test manager, the developers, the test analysts and executors, operations staff, user representatives, technical support staff, data administration staff, and quality support staff.

For each testing person, specify the period(s) when the person is required.

In Master Teat Plan, list all teams (or engineers) that will be implementing this test plan. See table below. In "Primary responsibility" give a shorthand (1 sentence) summary of this team's role in this project. It may be that this information is already in the "Features to be tested" table. If so, reference it here.

Table 13: Roles and Responsibilities

Team	Manager/ Owner/Lead	Geo Location	Primary responsibility
Validation Engineering			
Power and Performance			
Certification			

Hiring needs

Quoting ISO 29119:

"Identifies specific requirements for additional testing staff that are necessary for the test project or test sub process. Specifies when the staff are needed, if they should be temporary, full or part time, and the desired skill set. These may be defined by contract and business needs.

Note: Staffing could be accomplished by internal transfer, external hiring, consultants, subcontractors, business partners, and/or outsourced resources."

External Test Resources

- Will you use any Test Houses for this project? Which Test House(s)? What tasks are done by the test house?
- What is the strategy for selecting tasks that are delegated to a Test House?
- How is the test house monitored for quality and efficiency?
- Who manages them?
- What HW needs to be provided to the 3rd party? Who will order, ship and track it? If it is expensive or confidential HW, this is a VERY BIG deal – make sure to work on it on time.
- How will bug reports be passed to/from a Test House?
- How will you pass new releases to the Test House?

3rd party IP providers

- What 3rd party components exist in this project?
- What tests do you plan to do on that IP?
- What is the partition of test work between your test team and the 3rd party?
- Were there any agreements with the 3rd party regarding incoming quality and testing? Are there gates to accept this component (acceptance tests? Other?).
- Who will review and approve these agreements? Who will review test content from the 3rd party? Who will monitor that the 3rd party complies with their commitments? Are there clearly defined communication channels? Names?
- What HW needs to be provided to the 3rd party? Who will order, ship and track it?
- How will bug reports be passed to/from a 3rd party?
- How will you pass new releases to the 3rd Party? How would you get new drops from them? Who owns integration and is the test team involved in such activities?

Skills / Training Needs

Anything the testers need to learn to be effective in testing the feature?

If this test plan covers more than one test level (e.g. Unit + Integration), you may want to list different needs for each test level.

If this test plan covers safety related features or code, the skill set needed by the testers include basic understanding of safety concepts and practices. See ISO 26262-2:2018 Section 5.4.4 .

Appendix A – Test Levels

The following explains what each test level means. The Template was built with these definitions in mind.

Unit tests

A Unit of code is a single function. Unit tests are isolating the function from all its dependencies by use of mocks and stubs that simulate the behavior of the dependencies. Using a unit-test framework or tool, the mocks and stubs are controlled to create the desired test conditions. This full control over the surrounding environment allows achievement of very high code-coverage metrics, as required by ISO 26262 for SR code.

The above is the classic definition of unit-test: testing in complete isolation. However, teams can define the size of a unit differently (e.g. a Class can be tested as one unit). Regardless to what your team defines as a "unit", the important part is the isolation from the rest of the system. Following this logic, if your tests are running on code that is installed on the target hardware (e.g. it is not isolated from the hardware), you are running Integration or System tests.

Integration tests

This test level is the hardest to define clearly. It can take many forms and the boundary between integration and system tests is fuzzy.

Case in point: You develop FW for an embedded processor that is part of a larger system which contains other processors and peripherals. You load the FW to the target HW, boot the system's OS and load all the other peripherals FW and drivers. You now have a full system and you are testing that the features of your FW are working as required.

What test level is it?

Some organizations call it Integration test: we test that our FW and HW integrates well into the full system. That it works well with shared resources and with other SW components that make up the system.

Other organizations call it System test – this is a black-box testing, using the complete FW and the target HW; it is not just checking that parts of the FW integrate well with each other.

Some possible suggestions to define what is covered by Integration tests:

- Tests done on simulators or emulators; for example, pre-silicon testing
- Continuous Integration tests (testing the new pieces of code work well with older code)
- Tests done at early stage of development, as the initial pieces of code are brought together and loaded to the HW
- Tests done on a new feature that is added to an existing system

The definition you chose will also define if integration test is a relatively short, one-time event done at an early stage of the project or a stage that continues to exist in parallel to System tests.

Note: The Integration Test Plan template was written with the above amorphic definition in mind. If in your organization "integration tests" are applied to the full FW, installed on the target HW and on the target platform the Master|Feature Test Plan may fit better. But in truth – there is little difference between the Integration and the Master|Feature templates. It's just that in proper places, the term "integration" is used.

System test

As mentioned for Integration, you may decide that since you only test your SW and HW, and not the whole product, you are never doing "system tests". You always do Integration tests. This definition works well with ISO 26262:Part6, Section 11 where the "vehicle" is seen as the "system".

A different approach is in ISO 26262 Part 4: according to that standard, once you integrate the HW and SW of a building block, it is already a "system". Multiple such systems are then integrated together to make a more complex system ("system of systems" – although ISO 26262 does not use this term explicitly). At each level, there is "System integration and test" step.

The template provided in this eBook targets predominantly the building-block level and not the whole system-of-systems. To avoid confusion, this template does not use the term "System Test plan". Rather, it uses "Master Test Plan" or "Feature Test Plan":

- **Master Test Plan** is a high level test plan that outlines the overall approach to testing the product your team develops (regardless if it is a complete product or just a building block in a larger system). Some organizations call this document a "Project Test Plan".
- **Feature Test Plan** is the next-level breakdown from the Master plan. The test strategy for each feature listed in the master test plan is outlined in detail. The feature is further broken into sub-features and describes the test strategy and test design of the sub-features.

Whatever your terminology, if your testing is black-box testing of the complete deliverable coming from your team, installed on the target HW, you will probably do OK if you use the "Master | Feature Test Plan" template to describe your test plans.

The template can probably be used as a good starting point for integration and system test plans for a system of systems. In such case, use the Master | Feature Test Plan section as well.

Acceptance tests

Acceptance tests are tests done to prove that the system satisfies the customer's formal acceptance criteria. The tests are conducted to enable the user, customer or other authorized entity to determine whether or not the system meets their requirements.

While the Template does not have an Acceptance Test section, since these tests are black-box tests targeting the system level, using the Master|Feature test plan section would, in most cases, provide a reasonable template.

Appendix B – Relevant International Standards

Some notes on international standards that are related to SW test, functional safety and the automotive market.

There are three relevant standards:

- ISO 29119: Software and systems engineering – Software engineering. Part 3 deals with Test Documentation and is of direct relevance to this template.
- A-SPICE: Automotive SPICE Process Assessment / Reference Mode. This standard defines a quality management system (QMS) to be implemented by SW suppliers to the automotive industry.
- ISO 26262: Road vehicles — Functional safety. Parts 6 (Product development at the software level) and part 8 (Supporting processes) are specifically relevant to SW testing.

ISO 29119

ISO 29119 defines a number of documents that are relevant to software testing. Two are relevant to mention in relation to the Test Plan Template:

- Organizational Test Strategy (OTS)
- Test Design Specification

The **Organizational Test Strategy** is a document that describes testing in your organization. In simple terms, the OTS explains "how our team does testing".

Since you insist… here is how the standard explains the OTS (ISO 29119-3, section 5.3):

> "The Organizational Test Strategy is a technical document that provides guidelines on how testing should be carried out within the organization, i.e. how to achieve the objectives stated in the Test Policy. The Organizational Test Strategy is a generic document at an organizational level that provides guidelines to projects within its scope; it is not project-specific."

In almost all organizations, much of the test strategy, test project management and other test activities do not change between projects. The OTS describes how these activities are done. Once your team has an OTS, much of the information expected to be detailed in a Test Plan is already documented in the OTS. It means you don't need to re-write it again in each project's Test Plan – you just reference the OTS.

The **Test Design Specification** identifies the features to be tested and the test conditions derived from the test basis for each of the features, as the first step towards the definition of test cases and test procedures.

I found that it is natural and with less overhead to combine the Test Plan and the Test Design documents – especially when writing a Feature Test Plan. The sections titled "Test Design Specification" at each test-level contains details of the information that ISO 29119 expect to see there.

ASPICE

ASPICE defines a quality management framework. It outlines a Process Reference Model and Process Assessment Model. One aspect that ASPICE covers is software testing. For SW test documentation, ASCIE relies in part on ISO 29119 for details. Specifically, to comply with ASPICE requirements for content in a Test Plan, one needs to follow the ISO 29119 definition of the Test Plan.

ISO 26262

ISO 26262 defines specific requirements for Unit test, Integration test and System test. These are on top of the general requirement that code is developed under a Quality Management system.

Since the automotive industry requires that suppliers implement ASPICE, one can see the ISO 26262 requirements as an added level of details on top of the ASPICE requirements. To avoid having to maintain two different templates, one for code that contains safety requirements and one for the rest of the code, the Template was created to support both.

For safety-relevant code, the template includes sections and support tables for collecting the data that ISO 26262 requires. The Guide includes specific explanations how to use the tables and how to provide the information required for ISO 26262 compliance.

EuroSTAR Huddle

EuroSTAR Huddle

Europe's Largest Selection of Software Testing Content.

1,250+ Blogs | 100+ eBooks | 200+ Webinars

The EuroSTAR team invite leading testing experts to share their knowledge with the community on Huddle. When you join Huddle you can access an unrivalled selection of resources across all the latest topics in software testing.

Expand your testing knowledge and join us for regular live webinars from prominent speakers and top contributors to the world of testing. Ask for help in the Huddle Forum and avail of our Huddle blog for the latest articles and trending topics in testing. Being part of EuroSTAR Huddle is an investment in your ongoing professional development and will give you added skills to help you achieve the very best in your career.

www.eurostarhuddle.com

Join The Community

Our Events

When you have enjoyed the online resources on Huddle, the training continues at our annual software testing events.

Experience the welcoming EuroSTAR Community. Be inspired by exceptional speakers sharing real life testing experiences. Try out the latest tools in the Expo and take advantage of the nonstop networking to connect with leading testing experts and upcoming innovators all in one place.

EuroSTAR Software Testing & Quality Conference

Celebrating 30 years of the EuroSTAR Community in 2022 - the longest running and largest software testing conference in Europe welcomes over 1,000 delegates every year.

Visit EuroSTAR Website

EuroSTAR Huddle on Tour

We also host a number of smaller conferences and events in different locations and bring the EuroSTAR Huddle live experience to partner events around Europe.